RL-TR-92-309
Final Technical Report
November 1992

CHIP-LEVEL TESTABILITY REQUIREMENTS GUIDELINES

Research Triangle Institute

James W. Watterson, Mark Royals, Nick Kanopoulos

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

Rome Laboratory
Air Force Materiel Command
Griffiss Air Force Base, New York

This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RL-TR-92-309 has been reviewed and is approved for publication.

APPROVED:

WARREN H. DEBANY, JR.
Project Engineer

FOR THE COMMANDER

JOHN J. BART, Chief Scientist
Reliability Sciences

If your address has changed or if you wish to be removed from the Rome Laboratory mailing list, or if the addressee is no longer employed by your organization, please notify RL (ERDA) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	November 1992	Final Dec 90 - Oct 91

4. TITLE AND SUBTITLE

CHIP-LEVEL TESTABILITY REQUIREMENTS GUIDELINES

5. FUNDING NUMBERS

C - F30602-90-D-0097,
PE - 62702F Task 3
PR - 2338
TA - QC
WU - 01

6. AUTHOR(S)

James W. Watterson, Mark Royals, Nick Kanopoulos

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Research Triangle Institute
Center for Systems Engineering
PO Box 12194
Research Triangle Park NC 27709

8. PERFORMING ORGANIZATION REPORT NUMBER

RTI/4968/01F

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Rome Laboratory (ERDA)
525 Brooks Rd
Griffiss AFB NY 13441-4505

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

RL-TR-92-309

11. SUPPLEMENTARY NOTES

Rome Laboratory Project Engineer: Warren H. Debany Jr., RL(ERDA),(315) 330-2922
Prime Contractor: Rome Research Corporation

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

This report provides guidelines and rationale that will assist system developers in specifying consistent, necessary, and achievable chip-level testability requirements. A bottom-up design procedure is recommended that emphasizes the importance of chip testability estimation and built-in-test evaluation during chip design to minimize system life-cycle cost. A comprehensive set of cost-related testability attributes was developed, which address test generation, test application, and fault isolation and repair. This set of attributes was then prioritized on the basis of usefulness and estimation cost. A set of chip-level design-for-testability and built-in-test techniques were selected from open literature sources. Preliminary analyses of eleven of the more promising techniques considered nine criteria in the areas of: impact on testability, cost, and applicability to standard design practices. Three of the techniques were subjected to in-depth analyses that included the results of case studies. The techniques considered in the in-depth studies were: Circular Built-In-Self-Test, LSSD On-Chip Self-Test (LOCST), and CrossCheck.

14. SUBJECT TERMS

Testability, design-for-testability, built-in-test, fault coverage, reliability, availability, life-cycle cost

15. NUMBER OF PAGES

172

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

ACKNOWLEDGEMENTS

This report was prepared for the Rome Research Corporation under Contract No. RCC-SC-90-0097-04. The work was performed at the Research Triangle Institute by J.W. Watterson, Mark Royals, Nick Kanopoulos (Project Manager), and Ingrid Agolia (Secretary), under the technical direction of Dr. Warren Debany (RL).

Contents

List of Figures

List of Tables

EVALUATION

This is the final technical report for the task, "Chip-Level Testability," which was part of a Rome Laboratory program titled "Chip-Through-System Testability." The goals of this program include allocation of system-level testability requirements to subsystems, development of guidelines for the use of test bus standards (such as the IEEE Std 1149.1 Boundary Scan Bus), and the establishment of chip-level testability measurement and built-in-test requirements that would permit the bottom-up implementation of testable systems. The latter goal is the subject of this technical report.

This report outlines the fundamental considerations, decisions, and procedures involved in providing the building blocks of a highly-testable electronic system. A testable system is composed of testable components in testable configurations; both top-down testability allocation procedures and bottom-up testability implementation procedures dead-end when low-level testability features are not exploitable at higher levels.

Most technical discussions of "testability" do not bother to define the term. Instead, testability is often talked about in generic and abstract terms that are devoid of any relationship to the solution of engineering design, manufacture, and support problems. This report will make such discussions indefensible in the future. A complete hierarchy of testability-related factors, measures, and cost function mappings is presented here. The relationships between testability and its concommitant design penalties have been extensively discussed elsewhere; this report extends that approach by discussing in clear fashion the competitive nature of the testability requirements themselves.

Even more important than the exhaustive list of testability attributes is the culling-down of the list to a relatively small set of testability attributes that possess two properties: they are both *useful in practice* and *practical to obtain*. Thus, a set of testability attributes are presented that form a reasonable basis for the selection of achievable and measurable testability requirements. These are *chip-level testability requirements* that address *system-level concerns*.

To satisfy stringent testability requirements appropriate enabling mechanisms must be provided. Design-for-testability (DFT) and built-in-test (BIT) techniques were analyzed and characterised according to both their impact on testability attributes and their design penalties. The analyses were based on case studies that involved redesigns of sample circuits. Eleven techniques were considered in some depth, and three were characterised in detail.

This report provides a detailed set of guidelines for the selection of chip-level testability requirements. It provides a balanced and authoritative discussion of the DFT and BIT techniques that are needed to satisfy those requirements. It will be useful to System Program Offices, project engineers, and design and test engineers.

WARREN H. DEBANY JR., PH.D.
Project Engineer

1. INTRODUCTION

The objective of this research effort is to establish guidelines that will assist system developers in specifying consistent, necessary, and achievable chip level testability requirements. This program is an extension of earlier research on testability measurement and BIT evaluation [1,2,3], where BIT techniques were investigated and tools/techniques identified for estimating the cost of chip level testability. To this end, a bottom-up design procedure is recommended herein that emphasizes the importance of chip testability estimation and Built-In Test (BIT) evaluation during chip design to achieve the goal of minimizing system life-cycle cost.

A relationship between testability, BIT, and life-cycle cost is established by noting that an item's availability, reliability, maintainability, and life-cycle cost are of primary importance at the system level when determining the overall worth of a digital system. These interrelated parameters are further defined as follows:

1. Availability is the fraction of time a system is available for use,
2. Reliability is the conditional probability that the system is operating properly at time $t > 0$, given that the system is operational at t=0,
3. Maintainability relates to the ease with which a system fault can be detected, isolated, and repaired/replaced, and
4. Life-Cycle Cost includes:
   - Development
   - Manufacture
   - Installation
   - Operation
   - Maintenance
   - Replacement.

Observing these definitions, it is evident that one should attempt to minimize life-cycle cost while maximizing availability to increase the worth of the system.

The three "ilities" and life-cycle cost are related since life-cycle cost is related to reliability and maintainability, which are in turn functionally related to system availability. Considering a repairable system with a mean fault cycle, the inherent availability $A_i (0 < A_i < 1)$ is given by

$$A_i = \frac{MTTF}{MTTF + MTTD + MTTR} \tag{1.1}$$

where

MTTF = Mean Time To Failure
MTTD = Mean Time To Detect
MTTR = Mean Time To Repair

This equation reveals that the inherent availability of a system can be increased by reducing MTTD and MTTR. Such reductions in MTTD and MTTR can be achieved by incorporating cost-effective testability into a system during design and development of the system. In an earth-based environment, testing can be performed through use of Automatic Test Equipment (ATE) or BIT. Alternatively, in a space-based environment BIT is the logical choice for performing system tests.

When considering the cost-effectiveness of chip testability and the selection of BIT, two questions arise are: "What is the cost of testability?" and "Which chip level BIT techniques are appropriate?" This program addresses these questions by performing three tasks as follows:

Task 1 - Develop a Theory of Testability Measurement

Task 2 - Assessment of BIT/DFT Techniques

Task 3 - Develop Chip Level Testability Guidelines

Tasks 1 and 2 are closely related to the work performed by RTI under RL contract No. F30602-87-C-0105 [1,2,3].

Task 1 is accomplished by performing three subtasks. Subtask 1.1 consists of developing a Testability Attribute Set (TAS) consisting of testability factors and

measures for chosen attribute categories. The attribute categories are established by associating testability and the cost of testing to the cost-related areas (a) test generation, (b) test application, and (c) fault isolation and repair. Five testability attribute categories are identified that encompass these three areas, and testability factors and measures are identified for each attribute category. Testability factors (gate count, package count, fan-out, fault count, etc.), which are determined primarily from a proposed design, are indirectly related to the cost of testing a device. Testability measures (difficulty of obtaining tests, test set statistics, etc.) are quantities that are directly related to test cost.

The testability factors and measures identified in Subtask 1.1 can be combined with appropriate cost measures to compute meaningful estimates of test cost provided they can be estimated with a known confidence interval. In this regard, Subtask 1.2 consists of evaluating existing Testability Measurement Techniques (TMTs). The purpose of Subtask 1.2 is to assess the capabilities and limitations of testability tools/techniques in providing quantified measures of an item's testability and in assessing their potential for estimating the costs associated with testing of digital electronic systems. In Subtask 1.3, the TAS from Subtask 1.1 is prioritized and evaluated according to the TMT results of Subtask 1.2, resulting in the development of a Feasible Testability Attribute Set (FTAS). The FTAS is a subset of the TAS that is derived by considering practicality and estimation cost. The FTAS is then combined with TMT to establish a Testability Measurement Technique Set (TMTS) which can be used to estimate each of the attributes in the FTAS.

Task 2 is the identification and assessment of DFT/BIT techniques that may be attractive for use in digital systems. The two subtasks of this task are: (Subtask 2.1) systematic enumeration of currently available DFT/BIT techniques and their relative order of importance as judged by their relevance to digital system design; (Subtask 2.2) development of a design penalty set which consists of generic factors associated with the addition of DFT/BIT that adversely impact system design.

Task 3 is to develop chip level testability guidelines that incorporate the TMTS from Task 1 and the DFT/BIT techniques from Task 2. These guidelines, contained in this report, recommend a bottom-up procedure for assessing and validating the testa-

bility of a design during various stages of system development. This procedure will aid system developers in specifying cost-effective chip level testability requirements.

In this report it is assumed that the system being developed contains only digital chips. Hence, the guidelines are not directly applicable to hybrid chips that contain both analog and digital circuitry. Recognizing that components of new systems (especially above the chip level) may contain both analog and digital circuitry, the information contained herein is but one step toward developing a unified approach to specifying testability requirements for analog/digital components at all levels of the system hierarchy.

2. SPECIFICATION OF TESTABILITY REQUIREMENTS

Testability is viewed as a design characteristic that enhances system availability by contributing to fault detection and fault isolation, and thus system repair. The observed goal of testability is to reduce system life-cycle cost. Since there is significant up-front cost associated with introducing testability into a new design, the question then arises as to what procedure should be used to specify necessary and sufficient testability requirements and thereby obtain cost-effective testability.

A preliminary step toward specifying lower level testability requirements is the establishment of system testability requirements which, in many cases, are not stated explicitly. They can be derived by noting that they are functionally related to the given system requirements. Some examples of system requirements that are closely related to testability requirements are:

(a) Reliability requirements (MTTF, etc.)

(b) Maintainability requirements (MTTD, MTTR, etc.)

(c) Functional requirements (throughput, etc.)

(d) Physical requirements (size, weight, power drain, etc.)

Component testability requirements (chip, board, etc.) can then be established that satisfy the system testability requirements. Two approaches to specifying testability requirements, the top-down approach and the bottom-up approach, are briefly explored in the following paragraphs.

2.1. Top-down Approach (Alternative #1)

The top-down approach to specifying testability requirements is an iterative procedure that consists of an initial allocation of system testability requirements to lower levels as follows:

1. Allocate system testability requirements to subsystems,

2. Allocate subsystem testability requirements to modules,

3. Allocate module testability requirements to boards, and

4. Allocate board testability requirements to chips.

The distribution of testability requirements to lower levels would not necessarily be uniform. Those components that are known to be easy to test could be allocated more stringent testability requirements to allow a reduction of testability requirements on other low level components that are more difficult to test. The probability of fault occurrence in an item, which is influenced by item complexity, technology, environment, etc., should also be considered when deviating from a uniform distribution of testability requirements to lower level components. Unfortunately, no method is available for obtaining an accurate estimate of probability of fault occurrence for a chip that has not been designed.

As a simple example of the top-down approach, assume the system fault coverage requirement is 0.99 for detectable single stuck-at faults. This system fault coverage requirement can be satisfied by placing a fault coverage requirement of 0.99 on every lower level component, including component I/O and the interconnections between components (uniform distribution of fault coverage throughout the system).

After the initial allocation of testability requirements to lower levels within the system, development could proceed by

1. Designing the lower level components (with chosen testability features),

2. Reallocating system testability requirements when specified low level testability requirements are too stringent (hardware/time required to meet low level testability requirements cannot be satisfied),

3. Repeating steps 1 and 2 until lower level testability requirements are satisfied,

4. Verifying that system testability requirements are satisfied.

This top-down approach to testability allocation can undoubtedly be used to design testable systems. However, such testability will not necessarily be cost-effective

in cases where new chip designs are involved since the top-down approach does not consider the cost of testability when allocating lower level testability requirements. Further, one cannot know what testability requirements are too stringent until preliminary chip designs (with chosen testability features) have been evaluated for testability. Hence, the top-down approach could result in inconsistent and unnecessary testability requirements.

2.2. Bottom-up Approach (Alternative #2)

The bottom-up approach to specifying testability requirements is also an iterative procedure that involves:

1. establishing functional requirements for lower level components (subsystem, module, board, chip) that are consistent with system functional requirements,
2. performing preliminary chip designs using judiciously chosen testability features that are appropriate for the chosen technology and chip architecture,
3. estimating cost/performance of chip level testability from preliminary chip designs,
4. estimating cost/performance of board/module/subsystem level testability by exploring alternative higher level testability features,
5. specifying testability requirements from results obtained in steps 3 and 4,
6. evaluating system testability from testability features chosen in step 5, and
7. repeating steps 2 through 6 until system testability requirements are satisfied.

Steps 3 and 4 of this bottom-up procedure focus on evaluating the cost/performance of testability alternatives which is necessary to assure that cost-effective testability is incorporated into the new design. In this way, the bottom-up approach will result in consistent, necessary, and achievable testability requirements at all levels of the system hierarchy.

A comparison of the two approaches to specifying testability requirements reveals that the top-down approach is directed toward finding *a set* of testability requirements at various levels of the system hierarchy that will satisfy system testability requirements, without regard to whether or not the requirements result in cost-effective testability. Alternatively, the bottom-up approach focuses on minimizing the cost of testability by incorporating the necessary and sufficient amount of testability at each level of the system hierarchy.

3. CHIP LEVEL TESTABILITY REQUIREMENTS GUIDELINES

3.1. Recommended Procedure

When formulating guidelines for setting chip level testability requirements, it is important to recognize that unachievable chip testability requirements are not meaningful and do not contribute to the goal of cost-effective chip testability. This being the case, the system developer should specify chip testability requirements that are achievable with known DFT/BIT techniques. This objective can be met by investigating the testability of a preliminary chip design (or designs) prior to setting chip testability requirements. Noting that the cost of testability (hardware, test time, etc) increases rapidly with the amount of testability incorporated into a chip design, the system developer should also establish chip testability requirements that are consistent and necessary to system testability requirements. Given these objectives (achievable, consistent, and necessary chip testability requirements), it is concluded that the bottom-up approach outlined in previous section is the only realistic procedure for establishing chip testability requirements.

A flow diagram of the recommended procedure for setting chip testability requirements is sketched in Figure 3.1. Here it is assumed that chip functional requirements have been derived from system requirements, and information on DFT/BIT is available to the system developer. The procedure shown in Figure 3.1 is essentially an iterative three-step procedure that involves

Step A: Establishing achievable testability requirements for all chips,

Step B: Developing higher level testability requirements,

Step C: Establishing that DFT/BIT is sufficient and necessary.

Associated with each step in the procedure for setting chip testability requirements is a set of interrelated issues that confront the system developer. Further discussion of these issues appears immediately below.

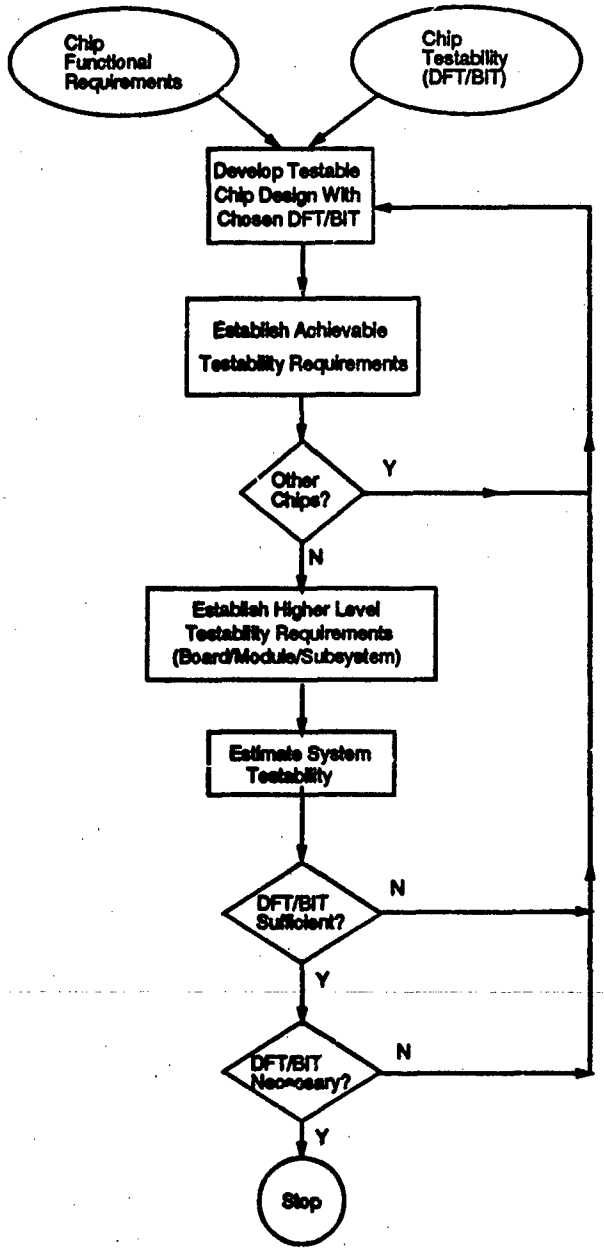

Figure 3.1. Bottom-Up Procedure For Establishing Testability Requirements

3.1.1. Setting Chip Testability Requirements

3.1.1.1. Preliminary Chip Design with DFT/BIT

The first step (step A) in the bottom-up procedure for setting chip testability requirements is the development of a preliminary chip design with judiciously chosen DFT/BIT. During this step many issues must be addressed to achieve the goal of designing a testable chip with cost-effective DFT/BIT. Design issues related to chip testability include

- Technology (TTL, NMOS, CMOS, GaAs, etc.)
- Chip Function
- Chip Architecture (structured, unstructured logic, etc.)
- DFT techniques (ad hoc, structured)
- Type of BIT (on-board test, off-board test)
- BIT test mode (off-line, on-line)
- BIT test set (deterministic, pseudorandom)
- BIT techniques (BILBO, error-detecting codes, etc.)
- Fault-tolerance (error-masking, self-repair)
- Chip Partitioning
- Silicon area allocated to DFT/BIT
- Chip simulation (transistor level, gate level, etc.)
- Design verification

It may not be apparent how some of the above issues impact testability. For example, technology is typically associated with performance. How is technology

related to testability? How are the other issues listed above related to testability? The following discussion of each issue is included to address these questions and also emphasize the importance of these issues to the system developer who strives to specify testability requirements.

- **Technology (TTL, NMOS, CMOS, GaAs, etc.)** – Technology is typically associated with performance since, in state-of-the-art designs, performance is one of the primary driving forces when identifying the technology to be used. Other issues typically considered when choosing a technology include scale of integration, power, compatibility with other technologies, and cost. Technology is also an important testability issue because each technology has unique failure mechanisms and failure modes. The system developer should be aware that chip failures, and our ability to model such failures and generate test sets to detect important faults, are related to technology [4,5]. Also, test application can vary widely in terms of test head (interface between chip and chip tester), ATE requirements, etc.

- **Chip Function** – Chip function must be considered when selecting a DFT/BIT technique since many important DFT/BIT are not universally applicable. Some examples of the relationships between DFT/BIT applicability and chip function are:

  - It is not cost-effective to use scan design (one of the DFT alternatives) on a RAM memory chip. Scan design is an attractive alternative for unstructured logic without embedded RAM/ROM memory.
  - The linear error-detecting codes (parity, Hamming, BCH, etc.) are applicable for detecting errors that occur on data paths or in memory. They are not generally applicable for detecting errors that occur while performing nonlinear operations (in a nonlinear operation the sum of two code words is not another code word).
  - Residue codes are applicable for concurrent detection of errors that occur when performing arithmetic operations (addition, multiplication, etc.).

The are not cost-effective for detecting errors that occur over data paths or in memory.

The system developer should be aware that chip performance may be degraded somewhat by the incorporation of DFT/BIT. Such degradation must be within acceptable limits for the chosen DFT/BIT. Otherwise, another DFT/BIT alternative must be considered.

- **DFT techniques (ad hoc, structured)** – The system developer must decide whether ad hoc or structured DFT techniques are appropriate in a given application. Ad hoc DFT (initialization, breaking global feedback paths, etc.) can be used to improve the testability of a sequential circuit. Alternatively, a structured DFT technique can be used to incorporate state-of-the-art testability into a new chip design. The use of a structured DFT technique is highly recommended for unstructured sequential logic.

- **Type of BIT (external test, built-in test)** – Test mode options include built-in test, external test, or a combination thereof. External test is carried out by applying test vectors from a source external to the system under test. Built-in self-test is accomplished by using test vectors that are stored within the system or generated within the system by circuitry such as the linear feedback shift register (LFSR). *Design for Testability* is compatible with any chosen combination of test mode options. The system developer should strive to select the test modes that are cost-effective for the given application. Some questions to be answered are:

  - Does the environment in which the system will be used lend itself to the use of portable test equipment? If not, built-in test should be used. For example, built-in test is the logical choice for performing tests on an aircraft controller while the aircraft is on a mission.
  - Do the system requirements for fault detection, fault diagnosis, and repair requirements permit one to choose between external and built-in test? The total time required to perform tests using built-in test is a small fraction of the time required when using external test equipment. If MTTD/MTTR

requirements are quite low, built-in test may be the only realistic test mode.

- Can existing portable test equipment perform the required chip tests? Will new portable test equipment have to be developed?

- **BIT test mode (off-line, on-line)** – On-line test, which can be either concurrent or background, is performed to detect faults that occur while the system is performing its intended function. Off-line test, performed when the system is not in operation, can be performed by either built-in or external test equipment. One question to be addressed by the system developer is whether of not the mission requirements require the use of on-line test. If on-line concurrent test is required, the error-detecting/correcting codes can be used to detect errors that occur on data lines, memory, or arithmetic operations. More generally, replication is a form of on-line concurrent built-in test that can be used to detect and mask errors that occur in a replicated component.

- **BIT test set (deterministic, pseudorandom)** – The system developer must decide whether to use deterministic test vectors, pseudorandom test vectors, or a combination thereof to test the chip. Deterministic test vectors are relatively expensive to generate and must be stored within the system to perform built-in test. Such test vectors can also be stored external to the system and applied by portable test equipment to achieve the desired tests. On the other hand, pseudorandom test vectors are readily generated at the time they are needed by clocking an LFSR which has been initialized prior to start of the test. The set of pseudorandom test vectors required to obtain a specified fault coverage is typically much larger than the required set of deterministic test vectors. Comparing pseudorandom test and deterministic test, the pseudorandom is less expensive to implement and is preferred when the fault coverage requirement can be achieved in an acceptable test application time interval.

- **BIT selection (BILBO, error-detecting codes, etc.)** – When selecting a BIT technique for use at any level of the system hiearchy, it is necessary to consider the following items to assure that BIT is optimum (or near optimum)

for the intended application:

- *Applicability of BIT techniques:* The chosen BIT technique must be an applicable test technique. For example, a Hamming code is not applicable for detecting errors in a parallel multiplier. A residue code is applicable for concurrent test on the multiplier.
- *Performance of applicable BIT:* The chosen BIT technique must achieve the testability goal. For example, assume the stated goal is to concurrently detect all single or multiple bit-errors that can occur over a data path. A parity code or Hamming code is not capable of achieving this objective. Replication is one approach to meeting this requirement.
- *Cost of applicable BIT:* The chosen BIT technique must be cost-effective. For example, assume the requirement is simply to detect single bit-errors that occur over a data path while the chip is performing its intended function. Any one of the linear codes (parity, Hamming, BCH, etc.) with a minimum Hamming distance $d_{MH} \geq 2$ could be used to satisfy the requirement, In this case the parity code is the appropriate choice since it requires significantly less hardware to implement.

- **Fault-tolerance (error-masking, self-repair)** – System availability can be increased by incorporating redundancy into the chips so that they are fault-tolerant. It is well-known that the vast majority of system failures are not permanent, but are instead transient in nature [6]. Error-detecting/correcting codes (data redundancy) can be used to mask errors that occur during system operation. Alternatively, a faulty component can be automatically replaced with a spare to achieve self-repair (hardware redundancy). Time redundancy in the form of data retransmission can also be used to implement fault tolerance. The use of fault-tolerance to implement testability is expensive since its implementation requires some form of redundancy (hardware, data, or time). This being the case, it should be used sparingly to satisfy system testability/reliability requirements in a cost-effective manner.

- **Chip Partitioning** – Chip partitioning into testable blocks of logic can be achieved by a carefully conceived ad hoc procedure or as an integral part of a structured design for testability technique. In either case, the primary objective of partitioning is minimize the number of test vectors required to control circuit nodes and observe the logic output to determine whether or not an error is detected.

- **Silicon area allotted to DFT/BIT** – How much silicon area on a chip should be allotted to DFT/BIT? This very difficult question is partially answered by noting that testability must be cost-effective to justify occupying chip area that would otherwise have been used for functionality. Silicon area should be allotted to those chip testability features that contribute to a reduction in life cycle cost of the system in which the chip is used. It is noteworthy that 28 VHSIC Phase I chip designs contained DFT/BIT, and the percentage of equivalent gates on a chip that were allotted to DFT/BIT ranged from 0.1% to 33%[7]. The relatively high value of 33% was used by IBM on the CMAC chip to obtain a fault coverage of greater than 99% of detectable single stuck-at faults.

- **Chip simulation (transistor level, gate level, etc.)** – Chip simulation is necessary during chip design to establish functionality and perform the test generation and fault simulation that is necessary to assure that the chip is testable with the chosen testability features.

- **Design verification** – Design verification is related to testability to the extent that functional test vectors generated early in the design process during functional simulation can contribute to the process of verifying that a gate level design is functionally correct.

Focusing on the objective of step A, which is to develop a preliminary chip design containing judiciously chosen DFT/BIT, it is evident that the above issues must be considered by the system developer when striving to select the optimum DFT/BIT.

3.1.1.2. Chip Testability Evaluation

Testability evaluation of the preliminary chip design is an essential step in the bottom-up approach to specifying achievable chip testability requirements that are ultimately necessary and sufficient. This step consists of obtaining relatively low-cost estimates of testability performance and cost, with meaningful bounds on the estimates, for the preliminary chip design. As indicated in Figure 3.2, this is accomplished by selecting a set of applicable testability attributes (factors and measures), obtaining estimates for the testability attributes during preliminary chip design (with bounds on the estimates), and using the estimated values of the testability attributes and DFT/BIT design penalties as a basis for specifying chip testability requirements. Issues related to performing testability evaluation on the preliminary chip design include the following:

- Testability factors (indirectly related to cost of testability)
- Testability measures (directly related to cost of testability)
- Methods for estimating testability measures (in polynomial time)
- Software tools to estimate testability measures
- DFT/BIT design penalties (chip area, I/O pins, etc.)

The use of practical testability measures and associated software tools to estimate chip testability in polynomial time permits (and even encourages) alternative DFT/BIT techniques to be considered during preliminary chip design. More detailed information on testability measures and tools/techniques is presented in Section 3.2.

The availability of information on the cost and performance of alternative DFT/BIT techniques is also essential to achieving the goal of specifying necessary and sufficient chip testability requirements. Such information contributes to the selection of optimum (or near optimum) DFT/BIT for a particular application. Further discussion of DFT/BIT design penalties appears in Section 4.2.

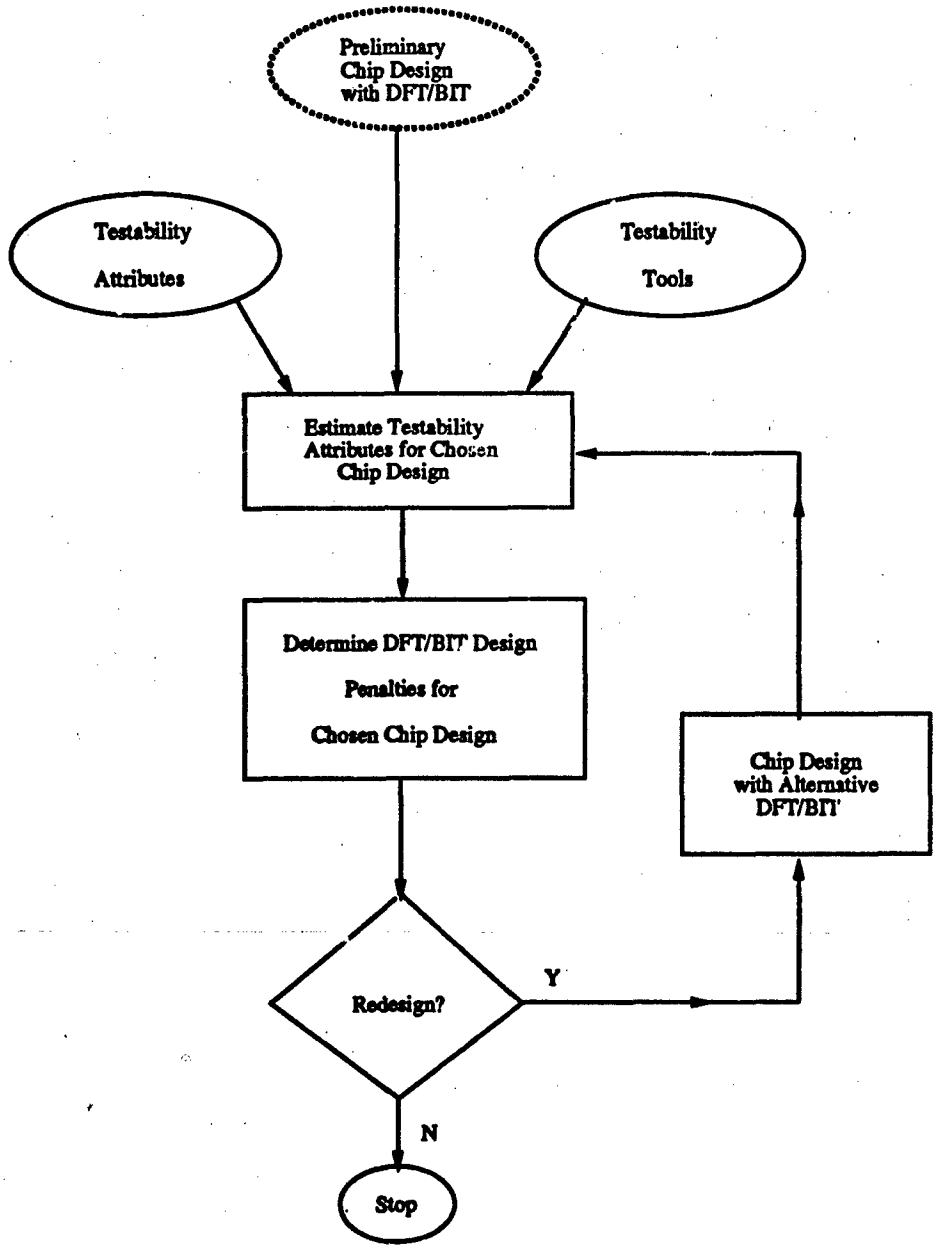

Figure 3.2. Testable Chip Design Cycle

3.1.2. Setting Higher Level Testability Requirements

While chip testability is the primary focus of this report, it is important to realize that higher level testability requirements (board, module, etc.) must also be established to achieve the testability goal of specifying cost-effective testability requirements for all system components. Issues related to Step B, establishing higher level (board, module, etc.) testability requirements, include the following:

- Achievable testability requirements (higher levels)
- Necessary testability requirements (higher levels)
- Consistent testability requirements (higher levels)

The question then arises as to what requirements are needed at the higher levels? This question is addressed by considering the block diagram representation of higher level digital system components shown in Figure 3.3. This sketch embodies the concept that

(a) a digital board containing only chips can be completely tested by testing the chips, the interconnections between the chips, and the board I/O,

(b) a digital module containing only boards can be completely tested by testing the boards, the interconnections between the boards, and the module I/O, and

(c) a digital subsystem containing only modules can be completely tested by testing the modules, the interconnections between modules, and the subsystem I/O, and

(d) a digital system containing only subsystems can be completely tested by testing the subsystems, the interconnections between subsystems, and the system I/O.

These observations about testing higher level components indicate that a system containing only digital chips can be throughly tested by testing

- all chips in the digital system,

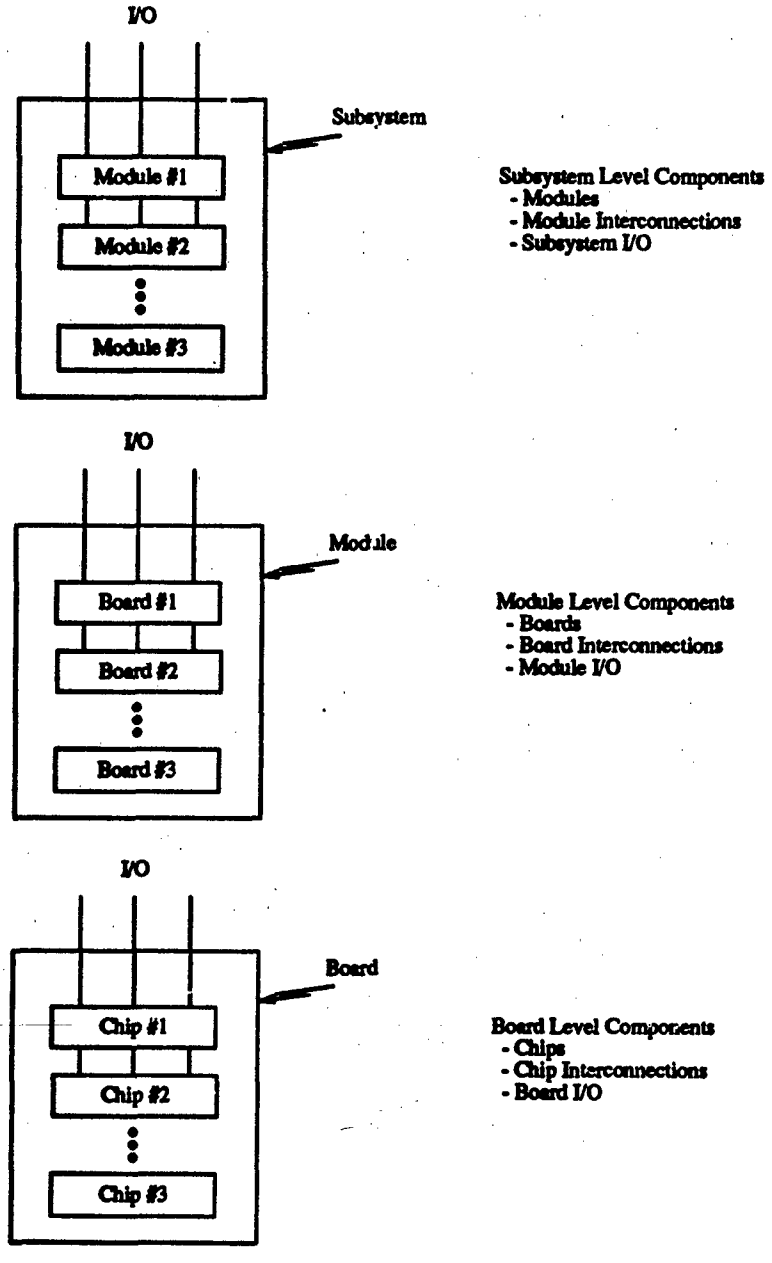

Figure 3.3. Higher Level Digital Components

- all interconnections between chips,
- all interconnections between boards,
- all interconnections between modules,
- all interconnections between subsystems, and
- the system I/O.

Hence, the seemingly difficult task of setting higher level testability requirements is reduced to setting testability requirements on interconnections between components (chips/boards/modules/subsystems) and the system I/O. The system developer should specify that interconnections between components be 100% tested for occurrence stuck-at-0/stuck-at-1 faults that may occur during manufacture or in the field. Tests for these common faults would also detect many bridging faults that are much more difficult to detect. Testability requirements on interconnections and component I/O can be satisfied by incorporating boundary scan at higher levels in the system design.

3.1.3. Testability Requirements Evaluation

As depicted in Figure 3.1, system testability is estimated (using the requirements specified for chips and higher levels) to establish that the chosen chip level DFT/BIT capability does in fact satisfy system testability requirements. This computation can also be used to answer the questions

(1) Is the chosen DFT/BIT sufficient?

(2) Is the chosen DFT/BIT necessary?

The chosen chip level DFT/BIT, which satisfies the chosen chip testability requirements, is sufficient when computations show that the DFT/BIT satisfies system testability requirements are satisfied. However, resulting system testability may exceed the system testability requirements, and in such case the chosen amount of chip level

DFT/BIT may not be necessary. When the computed system testability far exceeds the system testability requirements, it is appropriate to reduce the amount of chip level DFT/BIT which involves a redesign of one or more chips as indicated in Figure 3.1.

3.2. Chip Testability Estimation

With the rapidly increasing costs associated with the testing of digital electronic systems, much attention has recently been focused on the use of testability analysis and design for testability (DFT) techniques. Their goal is to cost-effectively assess and improve those design characteristics of an item that significantly increase the costs associated with testing.

Testability analysis should be applied early in the design process if it is to provide its maximum benefit. A testability analysis technique should provide an estimate of the difficulty of peforming a test-related task in such a way that the results can be used cost-effectively in making engineering tradeoffs. For example, there are two basic requirements for testability analysis methods used to assess an the difficulty of test generation for an item [8]:

1. The computational complexity should be significantly lower than that of test generation.
2. The calculated testability measures should reflect the potential difficulties for a specific strategy of test generation.

Many of the problems associated with the testing of digital electronic systems have been shown to belong to the class of NP-hard or NP-complete problems and thus, in the worst case, require computational effort that grows exponentially with the size of the problem. Still others, such as fault simulation, have been shown to be of polynomial complexity. It is obvious that problems of these complexities cannot be solved exactly in linear time. The goal of testability analysis is to use inexpensive and efficient heuristics to provide a reasonably close estimate (perhaps with bounds) of the desired measure(s).

3.2.1. Survey of Testability Tools/Techniques

3.2.1.1. Test Generation Tools and Techniques

One of the three important areas associated with testability is test generation[1]. There are presently no known practical methods for automatically generating tests for an arbitrary, general, sequential circuit. However, the inclusion of design-for-testability techniques allows one to treat a sequential item as being combinational for testing purposes. For the most part, this simplification can lead to substantial cost savings since test generation for combinational circuits is a well-understood procedure.

It is also important to note that testability analysis can be used to provide heuristical guidance in test generation algorithms. Many Automatic Test Pattern Generation (ATPG) tools use probabilistic or deterministic controllability/observability (C/O) measures as heuristics in their propagation (i.e., propagation of D or $\overline{D}$ from the fault site toward the primary output) and justification (consistency check after the propagation of D or $\overline{D}$ to the primary output) phases.

The D-Algorithm[9], PODEM[10], FAN[11], and GIPS[12] are four representative automatic test generation tools/techniques. Of these tools, the first three are algorithms which select a target fault and attempt to generate a test for it. This is done by attempting to justify a "good" value that is complementary to the "faulty" value on the target node, and propagating the effects of this value to an observable output. If conflicts occur in the node assignments, revisions are made in the assigned values until both objectives are either satisfied or all possibilities are exhausted. This implies that these three methods, if left unchecked, are guaranteed to find a test for the target fault if a test for that fault exists. The last tool, GIPS, is a low-cost method which attempts to generate a test for a fault but does not attempt to resolve any conflicts in the nodal assignments. If a conflict does occur (most often due to the occurrence of reconvergent fan-out), GIPS deliberately does not attempt to resolve it. Instead, it chooses an assignment for the node in conflict and continues to assign values to other nodes based on this choice until the primary inputs have been justified. This often results in other faults along the same sensitized path of the target fault being detected even though the initial target fault was not detected by the generated pat-

tern. However, GIPS does not guarantee that a test for a specific fault will be found even if the fault is non-redundant.

A list of some of the test generation tools and where they may be applied are listed in Tables 3.1 and 3.2[1]. More information about the capabilities and limitations of these tools is available in the detailed evaluations enclosed as appendices in an earlier report [3].

3.2.1.2. Fault Simulation Tools and Techniques

Traditionally, fault simulation has been used to verify the effectiveness of the generated test set by providing an indication of the percentage of faults that are detected by the applied test set. It is also used to create fault dictionaries to aid in the isolation of faults to a particular component or group of components. There are presently four different methods of fault simulation; serial, parallel, deductive, and concurrent. Serial and parallel fault simulation are rarely used nowadays due to their slow runtimes and their limited ability to handle multiple signal states [30]. Deductive and concurrent methods are faster, but they have large memory requirements due to their use of dynamic fault lists.

The basic principle behind fault simulation is to perform a simulation of the item under faulty conditions and compare the response to that of the fault-free simulation. If the presence of a fault produces an incorrect value on one or more of the outputs, the fault in question is said to be detected. Otherwise, it is undetected for that particular input pattern. Most conventional fault simulators use the stuck-at fault model, although many can also account for stuck-open faults and transistor shorts. Each of the four methods of fault simulation differ in the manner in which they process the faulty circuit. The serial approach invokes a complete simulation of the circuit with a single inserted fault. Thus, a total of m passes are required to complete the simulation where m is the total number of faults. Parallel fault simulation is different in that one can process as many as n faults in parallel where n is the number of bits in a word on the host computer. For an item with m faults, $[m/n]$ passes are needed to complete the simulation where $[m/n]$ is the smallest integer greater than or equal

Table 3.1. Test Generation Tools/Techniques

- Single Path Sensitization (IBM/AT&T)[13]
- DALG (D-algorithm; IBM)[9,14]
- SCIRTSS (Sequential Circuit Test Search System; University of Arizona) [15]
- TEST/80 (Testability and Test Generation Algorithm; Breuer & Friedman) [16]
- PODEM (Test Generation Algorithm; IBM)[10,17]
- FAN (Test Generation Algorithm; Osaka University)[11,18]
- HITS (Hierarchical Integrated Test Simulator; Naval Air Engineering Center)[19]
- TEGAS-5 (Test Generation and Fault Simulation; Calma Company)[20]
- ATWIG (Automatic TPG with Inherent Guidance; Siemens)[21]
- HITEST (Knowledge Based Test Generation System; Cirrus Computers)[22]
- RTG (Register Level Test Generator; AT&T)[23]
- SMART & FAST (Test Generator for Scan-Design Circuits; AT&T)[24]
- ESPRIT (Enhanced Statistical Production of Test Vectors; GE-Calma and BNR)[25]
- SOCRATES (Structure-oriented Cost-reducing Automatic Test Generation System; Siemens)[26]
- 9-V algorithm (9-Value Algorithm)[27]
- 10-V algorithm (10-Value Algorithm)[28]
- 16-V algorithm (16-Value Algorithm)[29]

Table 3.2. Test Generation Tool Comparison

Tool Name	Date	Fault Model	Type of Logic Network		
			Combinational	Sequential*	
				Syn.	Asyn.
DALG	1966	Stuck-At	X		
SCIRTSS	1977	Stuck-At	X	X	
TEST/80	1979	Stuck-At	X	X	X
PODEM	1981	Stuck-At	X		
FAN	1983	Stuck-At	X		
HITS	1983	Stuck-At	X	X	
TEGAS-5	1984	Stuck-At	X		
ATWIG	1984	Stuck-At	X	X	
HITEST	1984	Stuck-At	X	X	
RTG	1985	Stuck-At	X	X	
SMART & FAST	1986	Stuck-At	X		
ESPRIT	1986	Stuck-At	X		
SOCRATES	1987	Stuck-At	X		

* (Syn = synchronous, Asyn = asynchronous)

to the real number m/n [13].

For reasons previously mentioned, deductive and concurrent are the prevalent types of fault simulators in use today and are examples of event-driven simulators. The deductive approach consists of simulating the fault-free logic only and "deducing" the detectable faults from the "good" state of the item under test. A disadvantage of this implicit simulation method is that it requires the complete re-evaluation of a fault list when the fault list changes. On the other hand, the concurrent method simulates faults explicitly so that fault lists are modified in the portions of the item where the faulty behavior is different from the fault-free behavior. This allows for an even greater speedup in terms of simulation time at the expense of more required memory space.

Due to the limitations of fault simulation, recent efforts have focused on the development of approximate algorithms which are less memory intensive and have linear or near-linear run-times. Since the complexity of fault simulation is on the order of $O(n^2)$ to $O(n^3)$[31], where n is the number of gates, most of these approximate methods have advantages over conventional fault simulation. Proponents of these techniques point out that fault simulation is based on the use of a fault model which is itself only an approximation, thus the use of fault simulation to find a precise coverage is somewhat inefficient. Rather, it may be better to use approximate algorithms which provide relatively tight estimates and require lower computational effort than traditional fault simulation algorithms. A list of some of the currently available fault simulators and their corresponding techniques is shown in Tables 3.3 and 3.4.

3.2.1.3. Testability Measurement Tools and Techniques

There are several practical applications for testability analysis. Most of the its early uses were directed toward identifying those portions of an item which had poor testability. This information could then be used by the design engineer to modify the item in such a manner as to improve the testability of the deficient areas. The key to effective use of testability analysis in this application is the quality of the testability measures obtained from the analysis. It has been reported[43] that testability measures are often a poor indication of the testability of individual nodes. This results

Table 3.3. Fault Simulation Tools

- TEGAS-5 (Calma Company)[20]
- Daisy Fault Simulator (Daisy Systems Corporation)[20]
- VERIFAULT (Gateway Design Automation)[32]
- HILO (GenRad)[20]
- CADAT (HHB-Systems)[20]
- COFIS (Matra Design Systems)[20]
- LOGCAP II (Phoenix Data Systems)[20]
- THEMIS (Prime Computer)[20]
- BIFAS (Silvar-Lisco)[20]
- SILOS (SimuTec)[20]
- LASAR (Logic Automated Stimulus and Response; Teradyne Inc.)[20]
- Zycad Fault Evaluator (Zycad Corporation)[20]
- MegaFAULT (Daisy Systems Corporation)[33]
- FAUST (University of Illinois)[34]
- HITS (Naval Air Engineering Center)[19]

Related Tools

- STAFAN (Statistical Fault Analysis; AT&T)[35,36,37]
- TRUE (Testability Refinement by Undetectable Fault Estimation)[38,39]
- FAULT-BLASTER (Probabilistic fast fault grader; BNR)[40]
- AFS (Approximate Fault Simulator; Toshiba)[41]
- CPT (Critical Path Tracing; AT&T)[42]

Table 3.4. Fault Simulation Techniques

Tool	Simulation Technique(s)	Fault Model
TEGAS-5	Parallel	Stuck-At, Short
DAISY	Serial, Concurrent	Stuck-At, Open
VERIFAULT	Concurrent	Stuck-At
HILO	Parallel, Concurrent	Stuck-At, Inhibit Event, Short
CADAT	Concurrent	Stuck-At
COFIS	Concurrent	Stuck-At, Open, Short
LOGCAP II	Concurrent	Stuck-At, Open, Transistor Short
THEMIS	Concurrent	Stuck-At, Open, MOS Transistor Stuck-At
BIFAS	Parallel	Stuck-At
SILOS	(Proprietary)	Stuck-At, Open, Transistor Short
LASAR	Concurrent	Stuck-At, Open, Short
ZYCAD	Concurrent	Stuck-At, Open, Transistor Short
MegaFAULT	Concurrent	Stuck-At
FAUST	Concurrent	Stuck-At, Open, Transistor Short
HITS	Concurrent	Stuck-At

from the simplifying assumptions that are used in testability analysis to improve the speed and ease of use. Yet, there are indications that testability measures do provide a good indication of which sets of faults are more difficult to test. Thus, if the designer uses proper judgement and caution in his interpretation, testability measures can provide useful and meaningful information relating to the testability of the item.

Another application for testability measures is that of heuristics in guided automatic test pattern generation. The controllability/observability (C/O) cost measures are used to provide an indication of which lines are easier to control or observe when attempting to justify a value on a specific node or propagate the value to an observable location. Several studies[44,45,46] have demonstrated that C/O measures can reduce the time and cost associated with deterministic test generation. More recently, testability analysis has been used to assess the random pattern testability of an item. Probabilistic measures relating to the detectability of each fault can be obtained from an item and can be used to predict the fault coverage for a specified test length. Since deterministic test generation can be very expensive even for purely combinational circuits, an indication of whether an item can be satisfactorily tested with random patterns can result in significant cost savings.

Most testability analysis algorithms operate by parsing a topological or structural description of the item and then by providing a quantified measure related to the item's intrinsic testability characteristics. There are presently four basic types of testability analysis tools available[47]: fault detectability tools, simulation-based tools, heuristic scoring tools, and nodal dependency tools. Fault detectability tools can be further divided into deterministic and probabilistic methods. Tables 3.5 and 3.6 contain a list of some of the currently available tools as well as some information regarding their level of applicability. More detailed evaluations of SCOAP, PREDICT, CPT, and STAFAN can be found in [3].

Table 3.5. Testability Measurement Tools/Techniques

CHECKLIST TOOLS

- CODMOD (Consolla and Danner Model; RADC PCB checklist)[48,49,50]
- MILMOD (Military Model; MIL-STD-2165)[49,50,51]

CONTROLLABILITY-OBSERVABILITY TOOLS

Deterministic:

- TESTSCREEN (Testability Analysis Program; Sperry Research Center)[52]
- TMEAS (Testability Measurement Program; AT&T)[53]
- SCOAP (SANDIA Controllability/Observability Program; SANDIA Laboratories)[54,55]
- ITFOM (Inherent Testability Figure of Merit; Sperry-Univac Corporation)[56]
- CAMELOT (Computer-aided Measure for Logic Testability; Cirrus Computers)[57]
- ITTAP (Interactive Testability Analysis Program; ITT)[58]
- COMET (Controllability and Observability Measure for Testability; United Technologies)[59]
- VICTOR (VLSI Identifier of Controllability, Testability, Observability, and Redundancy; University of California)[60]
- COPTR (Controllability-Observability-Predictability-Testability Report; Calma)[61]
- HECTOR (Heuristic Controllability & Observability Analysis; Siemens)[62,46,12]
- CAFIT (Computer Aided Fault Isolation/Testability; NOSC)[63,64]
- A Calculus of Testability at the Functional Level (S. Takasaki; N on Electric)[65]

Probabilistic:

- COP (Controllability/Observability Program; BNR)[66]
- PREDICT (Probabilistic Estimation of Digital Circuit Testability; AT&T)[67]
- PROTEST (Probabilistic Testability Analysis; University of Karlsruhe)[68]
- ENTROPY (Information Theory Estimate of Testability)[69,70]

DEPENDENCY LOGIC MODELING TOOLS:

- STAT (System Testability Analysis Tool; Supersedes LOGMOD; DETEX)[71]
- LONGMOD (Longendorfer Model; Northrop) [49]
- STAMP (System Testability and Maintenance Program; ARINC)[50,72]

Other Testability Measurement Tools:

- TRI-MOD (Mission Effectiveness Testability Analysis; Giordano Associates)[73]
- HAT (Heuristic Advisor for Testability; University of Illinois)[74]

Table 3.6. Testability Tool Comparison

Tool	Date	Type of System		Level Within System			
		Analog	Digital	Chip	Board	Subsystem	System
CODMOD	1980		X		X		
MILMOD	1985	X	X	X	X	X	X
TEST SCREEN	1979		X	X	X		
TMEAS	1979		X	X	X	X	X
SCOAP	1980		X	X	X	X	X
ITFOM	1981		X	X	X	X	X
CAMELOT	1981		X	X	X	X	X
ITTAP	1982		X	X	X	X	X
COMET	1982		X	X			
VICTOR	1982		X	X			
COPTR	1983		X	X			
HECTOR	1984		X	X			
CAFIT	1988	X	X	X	X	X	X
COP	1984		X	X			
PREDICT	1985		X	X			
PROTEST	1985		X	X			
ENTROPY							
STAT	1988	X	X	X	X	X	X
LONGMOD	1982	X	X	X	X	X	X
STAMP	1984	X	X	X	X	X	X
TRI-MOD	1984	X	X	X	X	X	X
HAT	1985		X	X	X	X	X

3.2.2. Comprehensive List of Testability Measures

3.2.2.1. Testability Attribute Set Development

The goal of this endeavor is to develop a comprehensive list of well-defined testability attributes. Testability attributes consist of a set of testability factors and a set of testability measures. Testability factors (gate count, package count, etc.) are indirectly related to the cost of testing a device, while testability measures (node operations per fault, CPU seconds per detected fault, etc.) are directly related to test cost.

3.2.2.1.1. Testability Attribute Categories

Testability is a design characteristic that enhances system availability by contributing to fault detection and fault isolation, and thus system repair. More specifically, the incorporation of testability into a design accomplishes the following [75,76]:

1. Allows the status of a system (operable, inoperable, or degraded) or any of its subsystems to be determined in a confident and timely fashion, and
2. Facilitates fault isolation to a replaceable unit, and thereby contributes directly to system repair.

Some interrelated quantities that influence testability are (1) failure mechanisms for the item being considered, (2) chosen fault population, (3) complexity of the item (number of equivalent gates), (4) architecture of the item (affects controllability and observability of nodes within the item), (5) number of test vectors required to satisfy fault detection/isolation requirements, (6) availability of test equipment required to perform fault detection/isolation tests on the item, and (7) how easy the test equipment is to use.

Observing the above definition of testability and the existence of interrelated quantities that influence testability, it is further recognized that the following design attributes directly impact testability:

1. Item architecture (chip/board/subsystem/system)
2. Methodology for partitioning system into testable units
3. Synchronous/asynchronous design
4. Chosen semiconductor technology
5. Presence/absence of interoperability design standards
6. Presence/absence of standardized maintenance interfaces
7. Presence/absence of design for testability techniques
8. Fault isolation methodology

By considering the above definition of testability and design attributes that directly impact testability, it is further recognized that the following three areas directly impact the cost of testability:

1. Test generation
2. Test application
3. Fault isolation and repair

Test generation consists of generating test vectors (or test sequences) and recording the associated "good circuit response" established by applying the test vectors (sequences) to the known good circuit. The cost of test generation for fault detection is primarily influenced by the fault detection requirements, the architecture, and the topology of the item being considered. Test generation for general combinational logic is known to be NP-complete [77], and cost-effective methods for generating test sequences for general sequential logic are not yet developed.

Test application for fault detection consists of applying test vectors to an item and analyzing the item response to detect the presence of a fault. When the system is located in a ground-based environment, portable ATE and/or low-cost off-line BIT

are typically used to apply test vectors (deterministic and/or pseudorandom) to an item and to analyze the item response. Alternatively, a system located in a space-based environment must rely on off-line BIT (e.g., signature analysis, BILBO) and concurrent BIT (e.g., error detecting/correcting codes) to detect the presence of a system fault. In either environment, the cost of test application is related to the specific design style/criteria used during system development.

Fault isolation consists of performing tests on a faulty system to isolate a fault to a line-replaceable unit so that the faulty LRU may be replaced. Thus, fault isolation test vectors (sequences) must be available in order to accomplish fault isolation. When test points are judiciously located within a system and are monitored during fault detection tests, the fault detection test vectors may also be used for fault isolation. The system architecture and the BIT techniques chosen during system development will greatly impact the cost of fault isolation. It is then evident that the cost of fault isolation and repair is heavily influenced by (a) item architecture, (b) design style/criteria, and (c) effectiveness of BIT.

This discussion suggests that the three important areas that impact testability can be related to the following five attribute categories:

(a) Difficulty of obtaining tests for fault detection (i.e., test vectors and/or test sequences),

(b) Test set length and fault coverage statistics,

(c) Adherence to a specific design style or design criteria,

(d) Difficulty of achieving fault isolation, and

(e) Effectiveness of BIT.

These categories are discussed further in the following section, and factors and measures are identified for each category. It should be noted that the five attribute categories are not disjoint; certain relevant testability factors and/or measures can appear in more than one category.

3.2.2.1.2. Testability Factors and Measures

Testability factors (gate count, package count, fan-out, fault count, etc.) are obtained from a proposed design. These factors are indirectly related to the cost of testing a device. Therefore, quantification of each testability factor is necessary if it is to be useful when estimating the cost of testability.

Testability measures proposed herein (difficulty of obtaining tests, test set statistics, etc.) are quantities that are directly related to test cost. For example, fault coverage can be defined as the conditional probability that a fault is detected, given that a fault has occurred [78]. Fault coverage is then a measure of the effectiveness of a test set to detect faults from a given fault population. The following is a comprehensive list of testability factors and testability measures for the five attribute categories [1]. The code used is "F" or "M" for factor or measure, "A" through "E" representing the category, followed by an integer. Associated with each measure is a representative set of units. In some cases the size of the circuit, etc., may be in "gates" or "faults" or some other meaningful unit. Fractions may be weighted by failure rates, gate counts, transistor counts, etc.

3.2.2.1.2.1. Factors and Measures for Category A

Category A is "difficulty of obtaining tests for fault detection." As pointed out earlier, the test generation problem is known to be NP-complete[77]. This means that, in the worst case, the CPU time required to compute a set of test vectors for single stuck-at faults in combinational logic increases exponentially as the number of logic gates and inputs increases linearly. The problem is even more challenging for sequential logic as no practical, cost-effective, test generation techniques are known for general sequential logic circuits. It is then evident that Category A will have a major impact on the cost of testability. The following is a comprehensive list of testability factors and testability measures for this category.

Factors for Category A (chip/board/subsystem/system levels)

(Category A: Difficulty of Obtaining Tests for Fault Detection)

(FA1) Number of items (gates/chips/boards/subsystems)

(FA2) Amount of embedded memory (e.g., ROM, RAM)

(FA3) Amount of fan-out

(FA4) Amount of reconvergent fan-out

(FA5) Number of feedback loops

(FA6) Difficulty of achieving circuit initialization

(FA7) Chosen fault population

(FA8) Number of detectable faults

(FA9) Number of undetectable faults

(FA10) Number of equivalence classes for detectable faults

(FA11) Number of faults in each fault equivalence class

(FA12) Signal probabilities

(FA13) Probability of detection for each fault

(FA14) Ease of controlling and observing internal nodes (e.g., number of node assignments to establish a logic value at a node from the primary input and observe the value at the primary output)

(FA15) Fraction of chip/board/subsystem/system tested by nonconcurrent BIT

(FA16) Fraction of nonconcurrent BIT that applies to scannable logic

(FA17) Fraction of chip/board/subsystem/system tested by concurrent BIT

(FA18) Number of test vectors required to evaluate capability of concurrent BIT

(FA19) Presence/absence of design for testability

(FA20) Presence/absence of test structures to facilitate a hierarchical test methodology

(FA21) Methodology used to partition a chip/board/subsystem/system for test

(FA22) Length of scan chains

(FA23) Amount of sequential logic without scan capability

(FA24) Number of test points

(FA25) Location of test points

(FA26) Number of I/O pins

(FA27) Amount of asynchronous logic

(FA28) Amount of available computing capability (for test generation)

(FA29) Availability of software tools (for test generation)

Measures for Category A (chip/board/subsystem/system levels)

(Category A: Difficulty of Obtaining Tests for Fault Detection)

(MA1) *Number of node operations per fault for combinational logic (node operations per fault)*

(MA2) *Number of node operations per test vector for combinational logic (node operations per test vector)*

(MA3) *Number of CPU seconds per fault for combinational logic (CPU seconds per fault)*

(MA4) *Number of CPU seconds per test vector for combinational logic (CPU seconds per test vector)*

(MA5) *Number of node operations per fault for unscannable sequential logic (node operations per fault)*

(MA6) *Number of node operations per test sequence for unscannable sequential logic (node operations per test sequence)*

(MA7) *Number of CPU seconds per fault for unscannable sequential logic (CPU seconds per fault)*

(MA8) *Number of CPU seconds per test sequence for unscannable sequential logic (CPU seconds per test sequence)*

(MA9) *Number of CPU seconds to compute optimal input line signal probabilities for random testing (CPU seconds per item)*

MA10) *Percentage of detectable faults in the item's fault population (faults per fault)*

MA11) *Percentage of detectable faults in the item's fault population with probability of detection below a specified threshold (faults per fault)*

MA12) *Percentage of item for which deterministic test vectors shall be generated (gates per gate)*

MA13) *Percentage of item tested with BIT that requires deterministic test generation (gates per gate)*

MA14) *Presence/absence of design for testability (pass/fail measure)*

MA15) *Presence/absence of test structures to facilitate a hierarchical test methodology (pass/fail measure)*

MA16) *Number of CPU seconds per test vector for fault simulation (CPU seconds per test vector)*

3.2.2.1.2.2. Factors and Measures for Category B

Category B is "test set length and fault coverage statistics." This category is closely associated with test generation as well as test application. Test set size is directly related to test cost since CPU time required to generate and fault simulate the tests increases with the length of the test set. In addition, storage for the deterministic test set and time required to apply the test set increases with size of the test set.

The following is a comprehensive list of testability factors and testability measures for category B.

Factors for Category B (chip/board/subsystem/system levels)

(Category B: Test Set Length and Fault Coverage Statistics)

(FB1) Total number of faults

(FB2) Detectable fault count

(FB3) Presence/absence of structural design-for-testability provisions such as partitioning for test, scan design, use of easily testable logic arrays, control over feedback loops during test, etc.

(FB4) Number and location of available I/O (including test points)

(FB5) Initialization methodology employed for sequential logic, i.e., master reset versus homing sequences, etc.

(FB6) Ease of controlling and observing internal nodes

(FB7) Number of sequential states in a design that does not have scan features

(FB8) Amount of asynchronous logic present

(FB9) Length of each scan chain if a scan technique is implemented

(FB10) Number of reconvergent fan-out nodes in a combinational network

(FB11) Probability of occurrence of each fault

(FB12) Strategies and algorithms used in test generation

(FB13) Number of fan-out-free subnetworks present in a combinational network

(FB14) Test set sizes of the fan-out-free subnetworks present in a combinational network

(FB15) Probability of detection of each fault

(FB16) Expected test length for each fault

(FB17) Gate fan-in counts

(FB18) Number of logic levels in a combinational network

(FB19) Sequencing used in input stimuli application

(FB20) Methods of error detection used at the outputs of an item

(FB21) Fault latency

(FB22) Error latency

(FB23) Node toggling counts obtained by logic simulation

(FB24) Number of primary inputs upon which each primary output of a combinational logic block is dependent upon (input widths of individual cones of logic)

(FB25) Extent of overlap of individual cones of logic

(FB26) Presence of uncontrollable feedback loops, and the number of states of the components in each uncontrollable feedback loop

Measures for Category B (chip/board/subsystem/system levels)

(Category B: Test Set Length and Fault Coverage Statistics)

(MB1) ***Fault coverage obtained when using a given test set (faults per fault)***

(MB2) ***Detectable-fault coverage obtained when using a given test set (faults per fault)***

(MB3) ***Number of deterministic test vectors required to obtain a specified detectable fault coverage (test vectors per item)***

(MB4) ***Number of deterministic test vectors required to detect a given subset of all detectable faults (e.g., critical faults with high probability of occurrence) (test vectors per item)***

(MB5) *Number of test vectors needed to detect a covered fault when using a given test set and test application strategy (test vectors per fault)*

(MB6) *Number of random/pseudorandom test vectors required to obtain a specified detectable-fault coverage (test vectors per item)*

(MB7) *Number of random/pseudorandom test vectors required to detect a given subset of faults with a specified level of confidence (test vectors per item)*

(MB8) *Whether or not a given subset of faults is detected, when using a given test set (pass/fail measure)*

(MB9) *Size of pseudo-exhaustive test set for a combinational logic circuit (test vectors per item)*

(MB10) *Number of random/pseudorandom test vectors applied until the incremental fault coverage per test vector block (of given size) drops below a given threshold (test vectors per item)*

(MB11) *Detectable-fault coverage obtained when the incremental fault coverage per test vector block (of given size) drops below a given threshold when applying random/pseudorandom test vectors (faults per fault)*

3.2.2.1.2.3. Factors and Measures for Category C

Category C is "adherence to a specific design style or criteria." This category is of primary importance to all of test generation, test application, and fault isolation and repair as shown earlier in Figure 2. This category emphasizes that testability must be considered during system design and development to produce a system that satisfies testability requirements. A comprehensive list of testability factors and testability measures for Category C is presented immediately below.

Factors for Category C (chip/board/subsystem/system levels)

(Category C: Adherence to a Specific Design Style or Criteria)

(FC1) Gate fan-in counts

(FC2) Ability to externally load registers (serial or parallel methods)

(FC3) Modularity of the design

(FC4) Presence/absence of wired logic AND/OR connections

(FC5) Length of scan chains

(FC6) Number of external clocks

(FC7) Presence/absence of redundant logic and/or nodes

(FC8) Interconnection of items (bus-oriented or point-to-point)

(FC9) Presence/absence of dynamic circuitry

(FC10) Presence/absence of counters or timers to implement delays (avoid one-shots)

(FC11) Number of levels in a combinational logic circuit

(FC12) Amount of sequential logic

(FC13) Presence/absence of special hardware to break feedback loops during test

(FC14) Number and complexity of ambiguity groups

(FC15) Ability to disable internal clocks during testing

(FC16) Presence/absence of chip bypass capability during hierarchical test

(FC17) Presence/absence of pull-up resistors on tristate buses or on "fixed" logic connections

(FC18) Capability of existing ATE to perform a desired test

Measures for Category C (chip/board/subsystem/system levels)

(Category C: Adherence to a Specific Design Style or Criteria)

(MC1) *Presence/absence of an initialization methodology (power-on/master reset, homing sequence, etc.) (pass/fail measure)*

(MC2) *Percentage of synchronous logic (gates per gate)*

(MC3) *Percentage of sequential logic that is scannable (gates per gate)*

(MC4) *Percentage of an item that is tested by either external methods or BIT, or both (gates per gate)*

(MC5) *Presence/absence of a hierarchical test structure (e.g., TM and ETM buses, maintenance controllers, boundary scan) (pass/fail measure)*

(MC6) *Presence/absence of DFT techniques such as module partitioning-for-test and/or the use of easily testable logic structures (pass/fail measure)*

(MC7) *Percentage of an item tested by concurrent BIT (gates per gate)*

(MC8) *Percentage of an item tested by nonconcurrent BIT (gates per gate)*

3.2.2.1.2.4. Factors and Measures for Category D

Category D is "difficulty of achieving fault isolation." To incorporate fault isolation capability into a system, one should consider the following steps during system design and development.

(1) Identify ambiguity groups of items,

(2) Select cost-effective test techniques (deterministic or pseudorandom) to be used to perform fault isolation tests,

(3) Determine where the test vectors are to be applied and where the test points are to be located,

(4) Compute test sequences capable of detecting the presence of a fault in an ambiguity group, and

(5) Develop procedures for analyzing test response as required to achieve fault isolation to one of the ambiguity groups.

A fault's ambiguity group consists of a set of possible "locations" where the fault may be present. The ambiguity group may be a set of gates, chips, boards, modules, or boxes; most often it consists of "replaceable units" At the present time, ad hoc procedures are used for identification of ambiguity groups, test point placement, and selection of test techniques.

In a controlled environment (where all inputs to an ambiguity group are controllable, all ambiguity group outputs are observable, and there are no feedback paths from another ambiguity group), the problem of generating tests for fault isolation is essentially the same as the test generation problem encountered when considering Category A. That is, it is NP-complete[77]. When several ambiguity groups are interconnected (i.e., several ambiguity groups of chips on a board) and all test vectors are to be applied from the board primary inputs, the test generation problem and the problem of analyzing test response is increased many-fold. Except for special cases, it is not cost-effective to perform fault isolation on a complex board by performing tests from the board I/O. One approach being considered by the test community for circumventing this extremely difficult problem on a digital board is to use chips with built-in self-test and provide boundary scan on each chip to permit testing of interconnections between chips and the board I/O.

Taking into account the many facets of the fault isolation problem, a comprehensive list of testability factors and testability measures for Category D is presented below.

Factors for Category D (board/subsystem/system levels)

(Category D: Difficulty of Achieving Fault Isolation)

(FD1) Number of items (chips/boards/subsystems/replaceable units)

(FD2) Amount of asynchronous logic

(FD3) Amount of feedback between chips/boards/subsystems

(FD4) Amount of uncontrollable feedback between chips/boards/subsystems

(FD5) Number of ambiguity groups (at each level)

(FD6) Number of states in each ambiguity group

(FD7) Number of overlapping ambiguity groups

(FD8) Number of ambiguity group I/O pins

(FD9) Fault count for each ambiguity group

(FD10) Probability of detection for each fault

(FD11) Ease of controlling/observing internal nodes

(FD12) Number of isolatable faults

(FD13) Number of faults that are not isolatable

(FD14) Number of test points

(FD15) Location of test points

(FD16) Presence/absence of hierarchical fault isolation methodology

(FD17) Amount of BIT

(FD18) BIT fault detection capability

(FD19) BIT false alarm rate

(FD20) Amount of available computing capability for fault isolation analysis

(FD21) Availability of software tools (for board/subsystem/system level fault isolation test generation)

Measures for Category D (board/subsystem/system levels)

(Category D: Difficulty of achieving Fault Isolation)

(MD1) *Number of node operations per isolated fault (node operations per fault)*

(MD2) *Number of node operations per fault isolation test sequence (node operations per test sequence)*

(MD3) *Number of CPU seconds per isolated fault (CPU seconds per fault)*

(MD4) *Number of CPU seconds per fault isolation test sequence (CPU seconds per test sequence)*

(MD5) *Ambiguity group size (items per ambiguity group)*

(MD6) *Percentage of detected faults that are isolated to an ambiguity group of specified size or less (faults per fault)*

(MD7) *Number of test vectors required to isolate an isolatable fault when using a given test set and fault isolation strategy (test vectors per fault)*

(MD8) *Number of test vectors required to achieve a specified fault isolation coverage when using a given fault isolation strategy (test vectors per item)*

(MD9) *Percentage of system with concurrent built-in test that provides for fault detection and fault isolation to an ambiguity group (gates per gate)*

(MD10) *Presence or absence of hierarchical fault isolation methodology (pass/fail measure)*

3.2.2.1.2.5. Factors and Measures for Category E

Category E is "effectiveness of BIT." In the increasingly complex digital systems currently being developed, and especially space-based systems where external test (using ATE) is not an option, BIT is an attractive alternative for detecting faults. In many instances BIT can also be used to isolate the faulty item. This being the case, it is essential that the effectiveness of BIT be measured to assure that system

testability requirements are satisfied. To this end, testability factors and measures for Category E are presented below.

Factors for Category E (chip/board/subsystem/system levels)

(Category E: Effectiveness of BIT)

(FE1) Total number of faults in logic covered by BIT

(FE2) Total number of faults detectable by BIT

(FE3) Presence/absence of design-for-testability

(FE4) Number and location of I/O (including test points)

(FE5) Initialization methodology employed for sequential logic

(FE6) Length of test sequences required to test sequential logic

(FE7) Amount of asynchronous logic present

(FE8) Probability of occurrence of each fault

(FE9) Probability of detection of each fault

(FE10) Detectability profile of the portion of the item tested by BIT

(FE11) Expected test length for each fault when using BIT

(FE12) Sequencing of input stimuli application by BIT

(FE13) Methods of error detection, logging, compaction, and reporting used by BIT

(FE14) Isolation strategies and algorithms employed by BIT

(FE15) Design of BIT circuitry

(FE16) BIT failure rate

(FE17) Fault latency

(FE18) Error latency

(FE19) Number of components in unbroken feedback loops

Measures for Category E (board/subsystem/system levels)

(Category E: Effectiveness of BIT)

(ME1) *Detectable-fault coverage obtained by using BIT (faults per fault)*

(ME2) *Whether or not BIT detects a given subset of the fault population (e.g., critical faults with high probability of occurrence) (pass/fail measure)*

(ME3) *Percentage of detected faults that are correctly isolated by BIT to an ambiguity group of specified size or less (faults per fault)*

(ME4) *Whether or not BIT correctly isolates a given subset of the detected fault population to an ambiguity group of specified size or less (pass/fail measure)*

(ME5) *Percentage of "Item Failed" indications provided by BIT that are due exclusively to BIT faults (BIT alarms per alarm)*

(ME6) *Number of test vectors needed by BIT to detect a covered fault when using a given test set and test application strategy (test vectors per fault)*

(ME7) *Number of test vectors needed by BIT to isolate an isolatable fault when using a given test set and fault isolation strategy (test vectors per fault)*

(ME8) *Reconfiguration time for fault-tolerant systems with self-repair capability (seconds per reconfiguration)*

(ME9) *System recovery time after reconfiguration (seconds per recovery)*

(ME10) *Completeness or quality of recovery measured in terms of performance and reliability of the recovered system over performance and reliability of the original system before failure (throughput rate per throughput rate and/or reliability of the recovered system over that of the original system)*

3.2.2.1.3. Summary of Measures

The comprehensive list of testability measures presented in the preceding section are summarized immediately below to provide the reader with insight into the types of measures that have been identified in each attribute category.

Category A: Difficulty Obtaining Tests for Fault Detection

- Time required to generate deterministic tests (MA1 through MA8)
- Time required to generate optimum input line signal probabilities for random testing (MA9)
- Percentage of detectable faults in item's fault population (MA10 and MA11)
- Design for testability measures (MA12 through MA15)
- Fault simulation time (MA16)

Category B: Test Set Length and Fault Coverage Statistics

- Fault coverage obtained from a given test set (MB1, MB2, MB8, and MB11)
- Number of deterministic test vectors required (MB3, MB4, MB5)
- Number of random/pseudorandom test vectors required (MB6, MB7, MB9, MB10)

Category C: Adherence to a Specific Design Style of Criteria

- Architecture related (MC2, MC3)
- Design for testability measures (MC1, MC4, MC5, MC6, MC7, MC8)

Category D: Difficulty of Achieving Fault Isolation

- Time required to generate fault isolation tests (MD1, MD2, MD3, MD4)
- Fault isolation coverage (MD6)
- Number of test vectors required (MD7, MD8)
- Design for testability measures (MD5, MD9, MD10)

Category E: Effectiveness of BIT

- Fault detection coverage (ME1, ME2)
- Fault isolation coverage (ME3, ME4)
- Fault alarm rate (ME5)
- Number of test vectors required (ME6, ME7)
- Self-repair/recovery (ME8, ME9, ME10)

3.2.3. Methods for Estimating/Evaluating Measures

As discussed earlier in section 3.2, testability estimation is attractive only when the cost of estimating a chosen measure is significantly less than the cost of actually incorporating testability into a new design or performing the test-related task (such as test generation or fault simulation). This implies that testability measures of interest are feasible, where a feasible measure is readily estimated (with bounds on the estimate). The question then arises as to which of the measures presented in section 3.2.2 are feasible. This question has been addressed by considering known methods for estimating the measures with results presented below in Table 3.7 [1].

Table 3.7. Methods for Estimating/Evaluating Measures (Continued)

MEASURE	METHOD	COMMENTS
MA1 MA2 MA3	Estimation of C/O (SCOAP) [54,55]	* Linear run-time * Combinational or synchronous sequential logic * Accurate for fan-out-free logic * No proven bounds on error for logic with reconvergent fan-out
MA4 MA5	Estimation of C/O (HECTOR) [12,46,62]	* Linear run-time * An extension of SCOAP * High correlation to the actual number of node assignments when using GIPS to generate tests * No proven bounds on error
MA6 MA7 MA8	Statistical approach (fault sampling and test generation)	* Maximum run-time is approximately proportional to sample size when a limit is placed on test vector generation time (or number of backtracks) per fault * Combinational or synchronous sequential logic * Confidence interval is established for the average value of each measure

Table 3.7: Methods for Estimating/Evaluating Measures (Continued)

MEASURE	METHOD	COMMENTS
MA9	PROTEST [68]	* Exponential run-time in the worst case * Estimates optimum values for signal probabilities * Combinational logic * No proven bounds on error if approximations are made
MA10	VICTOR [60]	* Linear run-time and memory requirement * Combinational logic * Lower bound on MA10
	CAFIT [78,14,63]	* Polynomial run-time and memory requirement * Combinational or synchronous sequential logic * Upper bound on MA10
	PROTEST [68]	* Polynomial run-time and memory requirement * Combinational logic * Upper bound on MA10

Table 3.7: Methods for Estimating/Evaluating Measures (Continued)

MEASURE	METHOD	COMMENTS
MA11	COP [66] PREDICT [67] PROTEST [68]	* Polynomial run-time and memory requirement * Combinational logic * No proven bounds on error * PREDICT can determine exact fault detection probabilities in exponential time
	Cutting algorithm [79]	* Polynomial run-time * Combinational logic * Lower bound on detection probability; hence, an upper bound on MA11. However, the lower bound on detection probability is invariably 0, so its usefulness is limited.
	STAFAN [35]	* Linear run-time and memory requirement * Combinational or synchronous sequential logic * No proven bounds on error

Table 3.7: Methods for Estimating/Evaluating Measures (Continued)

MEASURE	METHOD	COMMENTS
MA12 MA13	Gate Count	* Linear run-time * Exact values obtained
MA14 MA15	Determine from knowledge of chosen DFT/BIT techniques	* Pass/fail measures may be of value in an expert system environment
MA16	Statistical approach (test set sampling and fault simulation)	* Run-time and memory requirement less than for full fault simulation * Combinational or synchronous sequential logic * Confidence interval is established for an upper bound on MA16

Table 3.7: Methods for Estimating/Evaluating Measures (Continued)

MEASURE	METHOD	COMMENTS
MB1	STAFAN [35,36,37]	* Linear run-time and memory requirement * Combinational or synchronous sequential logic * No proven bounds on error
MB2	Fast Fault Grader [40]	* Linear run-time and memory requirement * Combinational logic * No proven bounds on error
	Critical Path Tracing [42]	* Nearly linear run-time and memory requirement * Combinational logic * Lower bound on MB1 or MB2
	Approximate Fault Simulator [41]	* Linear run-time and memory requirement * Combinational logic synchronous sequential logic * No proven bounds on error
	Fault Simulation	* Polynomial run-time * Memory requirement may be excessive when using concurrent fault simulation * Multiple levels of item hierarchy can be simulated * Exact value established for MB1 * Exact value established for MB2 if detectable fault set is known; otherwise, a lower bound on MB2
	Statistical approach (fault sampling and fault simulation)	* Run-time and memory requirement less than for full fault simulation * Multiple levels of item hierarchy can be simulated * Confidence interval for exact value of MB1 * Confidence interval for exact value of MB2 if detectable fault set is known; otherwise, a confidence interval for a lower bound on MB2

Table 3.7: Methods for Estimating/Evaluating Measures (Continued)

MEASURE	METHOD	COMMENTS
MB3	Test Counting [80,81,82]	* Linear run-time and memory requirement * Combinational or * Lower bound on MB3 or MB4 when detectable fault set is known; otherwise, no proven bound on error
MB4	Fan-out-free Subnetworks [83]	* Linear run-time and memory requirement * Combinational logic * Upper bound on MB3 or MB4; lower bound on MB3 if circuit does not have redundancy
	Known Results on required test set size for PLAs, RAMs, and other special circuits [84,85]	* Exact test set size in some cases * Upper or lower bounds on test set size in other cases
MB5	Critical Path Tracing [42]	* Nearly linear run-time and memory requirement * Combinational logic * No proven bounds on error
	Fast Fault Grader [40]	* Linear run-time and * Memory requirement * Combinational logic * No proven bounds on error
	Fault Simulation	* Polynomial run-time * Memory requirements may be excessive when using concurrent fault simulation * Multiple levels of item hierarchy can be simulated * Exact value established for MB5
	Statistical approach (fault sampling and fault simulation)	* Run-time and memory requirement less than for full fault simulation * Multiple levels of item hierarchy can be simulated * Confidence interval is established for the average value of MB5

Table 3.7: Methods for Estimating/Evaluating Measures (Continued)

MEASURE	METHOD	COMMENTS
MB6 MB7 MB10	Analysis using signal probabilities and fault detection probabilities -Cutting algorithm [79] -COP [66] -PREDICT [67] -PROTEST [68]	* Polynomial run-time * Combinational logic * Cutting algorithm determines lower bound on fault detection probability; hence, an upper bound on MB6, MB7, and MB10, and a lower bound on MB11 * COP, PREDICT (approximate version), and PROTEST have no proven bounds on error
MB11	Critical Path Tracing [42]	* Nearly linear run-time and memory requirement * Combinational logic * Upper bound on MB6, MB7, and MB10; lower bound on MB11
	Fault Simulation	* Polynomial run-time * Memory requirement may be excessive when using concurrent fault simulation * Multiple levels of item hierarchy can be simulated * Exact value established for MB6, MB7, MB10, and MB11 if detectable fault population is known; otherwise, an upper bound on MB6, MB7, and MB10, and a lower bound on MB11
MB8	Critical Path Tracing [42]	* Nearly linear run-time and memory requirement * Combinational logic * Worst-case (conservative) estimate
	Fault Simulation	* Polynomial run-time * Memory requirement may be excessive when using concurrent fault simulation * Multiple levels of item hierarchy can be simulated * Exact value established for MB8

Table 3.7: Methods for Estimating/Evaluating Measures (Continued)

MEASURE	METHOD	COMMENTS
MB9	Analysis using linear codes and the logic structure [86,87]	* Polynomial run-time * Combinational logic * Upper bound on MB9
	Algorithmic procedures [88,89]	* Polynomial run-time * Combinational logic * Upper bound on MB9

Table 3.7: Methods for Estimating/Evaluating Measures (Continued)

MEASURE	METHOD	COMMENTS
MC1 MC5 MC6	Determine from the final design of the system	* Pass/fail measures may be of value in an expert system environment
MC2 MC3 MC4 MC7 MC8	Gate Count	* Linear run-time * Exact value is determined

Table 3.7: Methods for Estimating/Evaluating Measures (Continued)

MEASURE	METHOD	COMMENTS
MD1 MD2	Estimation of C/O SCOAP [54,55] HECTOR [12,46,62]	* Linear run-time * Combinational or synchronous sequential logic * Accurate for fan-out-free logic * No proven bounds on error for logic with reconvergent fan-out
MD3 MD4	Statistical approach (fault sampling and test generation)	* Maximum run-time is approximately proportional to sample size when a limit is placed on test vector generation time (or number of backtracks) per fault * Combinational or synchronous sequential logic * Confidence interval is established for average value of each measure
MD5	Logic Model Analysis STAT [71] STAMP [50,72]	* Polynomial run-time * Combinational or synchronous sequential logic * Each item is associated with an ambiguity group * Assumes a perfect test set and hence yields a lower bound on MD5

Table 3.7: Methods for Estimating/Evaluating Measures (Continued)

MEASURE	METHOD	COMMENTS
MD6 MD7	Fault Simulation	* Polynomial run-time * Memory requirement may be excessive when using concurrent fault simulation * Multiple levels of item hierarchy can be simulated * True value is established for MD6 or MD7
	Statistical approach (fault sampling and fault simulation)	* Run-time and memory requirement less than for full fault simulation * Confidence interval established for average value of MD6 or MD7
MD8	Information Theory approach [90]	* Polynomial run-time * Combinational or synchronous sequential logic * No proven bounds on error
MD9	Gate Count	* Linear run-time * Exact value is established for the measure
MD10	Determine from the final design of the system	* Pass/fail measures may be of value in an expert system environment

Table 3.7: Methods for Estimating/Evaluating Measures (Continued)

MEASURE	METHOD	COMMENTS
ME1	STAFAN [35,36,37]	* Linear run-time and memory requirement * Combinational or synchronous sequential logic * No proven bounds on error
	Fast Fault Grader [40]	* Linear run-time and memory requirement * Combinational logic * No proven bounds on error
	Critical Path Tracing [42]	* Nearly linear run-time and memory requirement * Combinational logic * Lower bounds on ME1 (1)
	Approximate Fault Simulator [41]	* Linear run-time and memory requirement * Combinational or synchronous sequential logic * No proven bounds on error
	Fault Simulation	* Polynomial run-time * Memory requirement may be excessive when using concurrent fault simulation * Multiple levels of item hierarchy can be simulated * Exact value is established for ME1 if detectable fault set is known; otherwise, a lower bound on ME1(1)
	Statistical approach (fault sampling and fault simulation)	* Run-time and memory requirement less than for full fault simulation * Multiple levels of item hierarchy can be simulated * Confidence interval for exact value of ME1 if detectable fault set is known; otherwise, a confidence interval for a lower bound on ME1(1)

Table 3.7: Methods for Estimating/Evaluating Measures (Continued)

MEASURE	METHOD	COMMENTS
ME2	Critical Path Tracing [42]	* Nearly linear run-time and memory requirement * Combinational logic * Worst-case (conservative) estimate (1)
	Fault Simulation	* Polynomial run-time * Memory requirement may be excessive when using concurrent fault simulation * Multiple levels of item hierarchy can be simulated * Exact value for ME2 (1)
ME3	Fault Simulation	* Polynomial run-time * Memory requirement may be excessive when using concurrent fault simulation * Multiple levels of item hierarchy can be simulated * Exact value is established for ME3
	Statistical approach (fault sampling and fault simulation)	* Run-time and memory requirement less than for full fault simulation * Multiple levels of item hierarchy can be simulated * Confidence interval is established for ME3 (1)

Table 3.7: Methods for Estimating/Evaluating Measures (Continued)

MEASURE	METHOD	COMMENTS
ME4	Fault Simulation	* Polynomial run-time * Memory requirement may be excessive when using concurrent fault simulation * Multiple levels of item hierarchy can be simulated * Exact value is established for ME4 (1)
ME5 ME8	System simulation and analysis by employing simulated fault insertion	* Polynomial run-time * Multiple levels of item hierarchy can be simulated * Confidence interval is determined for the average value of each measure when fault population is sampled (1) * Under certain assumptions, an upper bound can be obtained for ME5
ME9 ME10	Failure modes and effects analysis	* No proven bounds on error

Table 3.7: Methods for Estimating/Evaluating Measures (Continued)

MEASURE	METHOD	COMMENTS
ME6	Critical Path Tracing [42]	* Nearly linear run-time and memory requirement * Combinational logic * Upper bound on ME6 (1)
	Fast Fault Grader [40]	* Linear run-time and memory requirement * Combinational logic * No proven bounds on error
	Fault Simulation	* Polynomial run-time * Memory requirement may be excessive when using concurrent fault simulation * Multiple levels of item hierarchy can be simulated * Exact value is established for ME6 (1)
	Statistical approach (fault sampling and fault simulation)	* Run-time and memory requirement less than for full fault simulation * Multiple levels of item hierarchy can be simulated * Confidence interval is established for the average value of ME6 (1)
ME7	Fault Simulation	* Polynomial run-time * Memory requirement may be excessive when using concurrent fault simulation * Multiple levels of item hierarchy can be simulated * True value is established for ME7 (1)
	Statistical approach (test set (test set sampling and fault simulation	* Run-time and memory requirement less than for full fault simulation * Confidence interval is established for the average value of ME7 (1)
		(1) Assumes ideal BIT error detection capability.

3.2.4. Recommended Testability Measures

Measure feasibility and usefulness are the criteria used in the following paragraphs for establishing a recommended set of measures and techniques for estimating chip level testability.

3.2.4.1. Feasible measures

A measure is deemed feasible if it can be estimated with computational effort that grows only polynomially (preferably linearly) with the size of the circuit. The feasibility of each measure is established by considering the entries in Table 3.7 and the associated references that describe the estimation methods. Measure feasibility results [1] are presented below in Tables 3.8 through 3.12 (one table for each testability attribute category). Observing these tables, a feasibility score of 1(CL) indicates that the measure is feasible for combinational logic, and a feasibility score of 1(SL) shows that the measure is feasible for synchronous sequential logic. In this regard, each measure in Table 3.7 with a feasibility score of 1(CL) and/or 1(SL) is regarded as a feasible measure.

3.2.4.2. Useful Measures

Measure usefulness is a primary consideration when identifying practical measures for estimating test cost. A qualitative indication of measure usefulness is established by considering the applicability of each measure and developing an associated usefulness score.

To this end, it is noted that test cost occurs during all phases of the system life cycle that include research and development, manufacture, and field. For example, test cost is incurred when test patterns are generated during research and development, when tests are applied during manufacture to verify that the item is good before shipping, and when maintenance tests are performed in the field.

Test cost is also functionally related to the system hierarchy (chip, board, etc.) and the associated test strategy. That is, a hypothetical module containing fifty chips

with sequential logic on some chips is typically very difficult to test completely (if possible) when testing is performed at the module I/O. Alternatively, it would be much less time consuming to thoroughly test the module by using a hierarchal test strategy such as (a) performing chip level tests, (b) testing interconnections between chips, and finally (c) testing the module I/O. A quantitative indication of measure usefulness is thus derived by considering the system hierarchy and the applicability of each measure to categories of the system life cycle. Measure applicability results[1] are presented below in Tables 3.8 through 3.12.

3.2.4.3. Recommended Measures

Table 3.8. Measure Recommendation (Category A)

Proposed Measure	Measure Feasibility and Usefulness		Measure Recommended (Yes/No)
	Feasible[1] (Yes/No)	Useful (Yes/No)	
MA1 thru MA4	Yes (CL)	Yes	Yes
MA5 thru MA8	Yes (CL) Yes (SL)	Yes	Yes
MA9 MA8	No (CL) No (SL)	No	No
MA10	Yes (CL) Yes (SL)	Yes	Yes
MA11	Yes (CL) No (SL)	No	No
MA12	Yes (CL) Yes (SL)	Yes	Yes
MA13	Yes (CL) Yes (SL)	Yes	Yes
MA14 and MA15	Yes (CL) Yes (SL)	Yes	Yes
MA16	Yes (CL) Yes (SL)	Yes	Yes

[1](CL = Combinational Logic; SL = Sequential Logic)

Table 3.9. Measure Recommendation (Category B)

Proposed Measure	Measure Feasibility and Usefulness		Measure Recommended (Yes/No)
	Feasible[1] (Yes/No)	Useful (Yes/No)	
MB1	Yes (CL) Yes (SL)	Yes	Yes
MB2	Yes (CL) Yes (SL)	Yes	Yes
MB3	Yes (CL) No (SL)	Yes	Yes
MB4	Yes (CL) No (SL)	No	No
MB5	Yes (CL) Yes (SL)	Yes	Yes
MB6	Yes (CL) Yes (SL)	Yes	Yes
MB7	Yes (CL) Yes (SL)	No	No
MB8	Yes (CL) Yes (SL)	No	No
MB9	Yes (CL) No (SL)	Yes	Yes
MB10	Yes (CL) Yes (SL)	No	No
MB11	Yes (CL) Yes (SL)	Yes	Yes

[1](CL = Combinational Logic; SL = Sequential Logic)

Table 3.10. Measure Recommendation (Category C)

Proposed Measure	Measure Feasibility and Usefulness		Measure Recommended (Yes/No)
	Feasible[1] (Yes/No)	Useful (Yes/No)	
MC1	Yes (SL)	Yes	Yes
MC2	Yes (CL) Yes (SL)	Yes	Yes
MC3	Yes (CL) Yes (SL)	Yes	Yes
MC4	Yes (CL) Yes (SL)	Yes	Yes
MC5	Yes (SL)	Yes	Yes
MC6	Yes (SL)	Yes	Yes
MC7	Yes (CL) Yes (SL)	Yes	Yes
MC8	Yes (CL) Yes (SL)	Yes	Yes

[1](CL = Combinational Logic; SL = Sequential Logic)

Table 3.11. Measure Recommendation (Category D)

Proposed Measure	Measure Feasibility and Usefulness		Measure Recommended (Yes/No)
	Feasible[1] (Yes/No)	Useful (Yes/No)	
MD1	Yes (CL) Yes (SL)	Yes	Yes
MD2	Yes (CL) Yes (SL)	Yes	Yes
MD3	Yes (CL) Yes (SL)	Yes	Yes
MD4	Yes (CL) Yes (SL)	Yes	Yes
MD5	Yes (CL) Yes (SL)	Yes	Yes
MD6	Yes (CL) Yes (SL)	Yes	Yes
MD7	Yes (CL) Yes (SL)	Yes	Yes
MD8	No (CL) No (SL)	Yes	No
MD9	Yes (CL) Yes (SL)	Yes	Yes
MD10	Yes (CL) Yes (SL)	Yes	Yes

[1](CL = Combinational Logic; SL = Sequential Logic)

Table 3.12. Measure Recommendation (Category E)

Proposed Measure	Measure Feasibility and Usefulness		Measure Recommended (Yes/No)
	Feasible[1] (Yes/No)	Useful (Yes/No)	
ME1	Yes (CL) Yes (SL)	Yes	Yes
ME2	Yes (CL) Yes (SL)	No	No
ME3	Yes (CL) Yes (SL)	Yes	Yes
ME4	Yes (CL) Yes (SL)	No	No
ME5	Yes (CL) Yes (SL)	No	No
ME6	Yes (CL) Yes (SL)	Yes	Yes
ME7	Yes (CL) Yes (SL)	Yes	Yes
ME8	No (CL) No (SL)	Yes	No
ME9	Yes (CL) Yes (SL)	No	No
ME10	Yes (CL) Yes (SL)	No	No

[1](CL = Combinational Logic; SL = Sequential Logic)

3.2.5. Examples

3.2.5.1. Estimating Test Generation Time

3.2.5.1.1. Statistical Approach to Evaluating Measures

Recognizing that test generation is very expensive for large blocks of combinational logic, it would be beneficial to estimate in polynomial time the cost of computing test vectors for combinational/sequential logic to achieve the goal of estimating the cost of testability. For example, consider measure MA4 which is defined as "the number of CPU seconds per test vector for combinational logic." There are no known polynomial time algorithms for estimating MA4 with exact bounds; therefore, it is reasonable to consider a statistical approach which results in a confidence interval for the measure[91]. After obtaining a confidence interval for MA4, we can multiply by Q (lower bound on required number of test vectors to completely test the logic) to establish a confidence interval for the lower bound on CPU time required to obtain a set of test vectors for detecting all detectable faults in the given combinational logic circuit.

3.2.5.1.1.1. Some Assumptions

The application of statistics to the estimation problem is accomplished by making the following assumptions:

1. Assume that each failure of interest can be modeled as a single stuck-at fault. With this assumption, the fault population consists of the set of all single stuck-at faults. The actual number of faults in the fault population is a function of a number of parameters that include (a) the chosen circuit, (b) the circuit model (transistor level, gate level, etc.), (c) the semiconductor technology, and (d) the types of failures being considered. Some of these faults may be redundant, and therefore undetectable by applying a test vector to the circuit primary input lines and observing the primary output lines.
2. Assume there are M detectable faults in given circuit.

3. Assume the given circuit has N detectable fault equivalence classes, denoted by the set $\{F_1, F_2, ..., F_i, ..., F_N\}$.

4. Assume each detectable fault equivalence class is equally likely to be chosen during test generation (the uniform probability mass function is suggested as an alternative to the actual function which is typically unknown).

5. Assume the chosen fault-dependent test generation procedure (i.e., one which targets specific faults and not a sensitized path) is capable of computing a test vector for each detectable fault. In general, there will be many test vectors for each fault, and the actual test vector generated will depend on the specific fault selected and the test generation algorithm.

6. Assume a representative fault f_i is selected from each fault equivalence class F_i to establish the fault set $\{f_1, f_2, ..., f_i, ..., f_N\}$.

7. Assume CPU time t_i is the test generation time for fault f_i. Each t_i is a function of parameters that include (a) the circuit being considered, (b) the test generation algorithm, and (c) the throughput of the computer being used. The CPU times that correspond to fault classes in the detectable fault population are contained in the set $\{t_1, t_2, ..., t_N\}$.

8. Each t_i in the set of CPU times is assumed to be a constant. That is, the algorithm used always follows the same steps when computing a test for fault f_i.

3.2.5.1.1.2. Confidence Interval Determination

Now consider an experiment on a given circuit that consists of randomly sampling the set $\{f_1, f_2, ..., f_i, ..., f_N\}$ and observing the computer CPU time required to generate each test vector. An outcome of the experiment, the CPU time required to generate a test vector, will be a finite (or countably infinite) number of computer clock cycles. Hence, the sample space is discrete since it contains a finite or countably infinite number of sample points. Points in the sample space are mutually exclusive, and

each point is called an elementary event. Each collection of like points in the sample space is referred to as an event to which a probability is assigned.

For example, let "number of CPU seconds per test vector for combinational logic" be the random variable, denoted by T. (Recall that a one-dimensional random variable x is a function that maps points in a sample space to points on the real x-axis). T is a discrete random variable since it is associated with a discrete sample space.

Associated with discrete random variable T is the probability mass function p(T). If p(T) were known, one could compute the exact mean μ, where μ is the average CPU time required to compute a test vector for the circuit being considered. Similarly, one could evaluate the exact variance σ_2, where σ_2 is a measure of the dispersion of p(T). More specifically,

$$\mu = \sum_i T_i \cdot p(T_i) \tag{3.1}$$

and

$$\sigma^2 = \sum_i (T_i - \mu)^2 \cdot p(T_i). \tag{3.2}$$

Unfortunately, p(T) is unknown for any given logic circuit, and for this reason precise values of μ and σ^2 cannot be determined.

An alternative approach is to use sampling theory and establish a confidence interval for the desired quantity μ. This can be accomplished by proceeding as follows:

1. Choose k samples from the set $\{t_1, t_2, ..., t_N\}$, with replacement, to obtain $(T_1, T_2, ...T_k)$,
2. View the resulting sample mean $\tilde{\mu}_k$ as a new random variable, and
3. Invoke the central-limit theorem which states that the sample mean $\tilde{\mu}_k$ is approximately normally distributed [92]. The approximation improves as sample size increases, and in the following discussion it is assumed that sample size k is large enough to assure that $\tilde{\mu}_k$ is for all practical purposes a normally distributed random variable.

Assuming a finite population of fault classes and random sampling with replacement, the expected value and variance of this new random variable are[93]

$$E(\tilde{\mu}_k) = \mu \tag{3.3}$$

and

$$Var(\tilde{\mu}_k) = \frac{\sigma^2}{k}. \tag{3.4}$$

Knowledge that the sample mean $\tilde{\mu}_k$ is approximately normally distributed suggests that we attempt to estimate its mean and variance.

The k values $\{T_1 T_2, ..., T_k\}$ can be established as follows:

1. Identify detectable fault equivalence classes for the given circuit,
2. Select a representative fault from each fault class to establish the set $\{f_1, f_2, ..., f_N\}$,
3. Randomly select a fault from the set $\{f_1, f_2, ..., f_N\}$,
4. Generate a test vector to detect the fault (an alternative fault may have to be selected if the algorithm backtrack limit is exceeded) and record the test generation time,
5. Repeat steps (3) and (4) as required to obtain $\{T_1, T_2, ..., T_k\}$.

A sample value of estimator $\tilde{\mu}_k$ (denoted by $\overline{\mu}_k$), which is an unbiased estimate of μ, is then computed by using

$$\overline{\mu}_k = \frac{1}{k} \cdot \sum_{i=1}^{k} T_i. \tag{3.5}$$

Observing (3.4), an estimate of the population variance σ^2 is needed to estimate the variance of the new random variable $\tilde{\mu}_k$. Noting that we are dealing with a finite set of N events and sampling is performed with replacement, an unbiased estimate of the population variance σ^2, denoted by S^2, is given by[93].

$$S^2 = \frac{1}{k-1} \cdot \sum_{i=1}^{k} (T_i - \overline{\mu}_k)^2 \tag{3.6}$$

An estimate for the variance of the normally distributed sample mean, denoted by $S^2_{\overline{\mu}}$, is then obtained by inserting (3.6) into (3.4) to obtain

$$S^2_{\overline{\mu}} = \frac{1}{k \cdot (k-1)} \cdot \sum_{i=1}^{k} (T_i - \overline{\mu}_k)^2. \tag{3.7}$$

Observing the results thus far, we can use (3.5) to obtain an unbiased estimate of the desired quantity μ. Assuming the continuous approximation of discrete mass function p(T) is normally distributed, $\tilde{\mu}$ is also approximately normally distributed even for small sample size. Then, for the case where $k > 30$, we can use (3.7) to obtain the following approximate confidence interval for μ (with a 1-α level of confidence)[93]:

$$\overline{\mu}_k - z_{\frac{\alpha}{2}} \cdot S_{\overline{\mu}} < \mu < \overline{\mu}_k + z_{\frac{\alpha}{2}} \cdot S_{\overline{\mu}} \tag{3.8}$$

where $z_{\alpha/2}$ is the point along the abscissa of the standard normal N(0,1) such that the area under the upper tail to the right of $z_{\alpha/2}$ is $\alpha/2$.

When $k < 30$, (3.8) can produce inaccurate results due to error in estimating the population variance σ^2. This, in turn, can result in a poor confidence interval. The problem associated with small sample size is overcome by introducing the Student t-distribution with k-1 degrees of freedom [92,93] to obtain the following confidence interval for μ (with a 1-α level of confidence):

$$\overline{\mu}_k - t_{\frac{\alpha}{2}} \cdot S_{\overline{\mu}} < \mu < \overline{\mu}_k + t_{\frac{\alpha}{2}} \cdot S_{\overline{\mu}} \tag{3.9}$$

The quantity $t_{\frac{\alpha}{2}}$ in (3.9) is defined to be the point along the abscissa of the t-distribution probability density function p(t) such that the area under the upper tail of p(t) to the right of $t_{\frac{\alpha}{2}}$ is equal to $\frac{\alpha}{2}$. That is, $P(t \geq t_{\frac{\alpha}{2}}) = \frac{\alpha}{2}$. Equation (3.9) has been observed to be highly reliable even when T is considerably different from the normal[93].

For example, assume a sample size k=30 is chosen for a given hypothetical circuit that has a total of N=200 detectable fault classes. Also assume that sample values for T_i, when inserted into (3.5) and (3.7), produce the estimates $\overline{\mu}_k = 4.770$ CPU seconds per test vector and standard deviation $S_{\overline{\mu}_k} = 0.120$. To obtain a 98% confidence interval for the true mean μ, refer to a table of values for the t-distribution with k-1=29 degrees of freedom and $\alpha = (1 - 0.98) = 0.02$. From[93] it is observed that $t_{\frac{\alpha}{2}} = 2.462$. Inserting these values into (3.9), the required interval (with 98% confidence) is

$$4.770 - (2.462) \cdot (0.120) < \mu < 4.770 + (2.462) \cdot (0.120) \tag{3.10}$$

or

$$4.475 < \mu < 5.065. \tag{3.11}$$

3.2.5.1.1.3. Estimating Sample Size

Sample size k must be chosen prior to performing a random experiment to obtain $\overline{\mu}_k$, an unbiased estimate for the population mean μ. Unfortunately, a precise sample size cannot be determined when the population probability mass function p(T) is unknown. However, a reasonable estimate for k can be established when information is available concerning the dispersion of p(T).

For example, assume p(T) is normally distributed with unknown mean μ and known variance σ_2. Focusing on the confidence interval for the sample mean, we are $100(1 - \alpha)$ percent confident that $\overline{\mu}_k$ deviates from the μ by less than E where[93]

$$E = (z_{\frac{\alpha}{2}}) \cdot \frac{\sigma}{\sqrt{k}} \tag{3.12}$$

This implies that k be chosen so that

$$k = (z_{\frac{\alpha}{2}} \cdot \frac{\sigma}{E})^2, \tag{3.13}$$

Exemplifying the use of (3.13), assume we are considering a normal process with unknown mean μ and known standard deviation $\sigma = 1.770$. Assume further that we want to know the sample size requi ed to be 95% sure ($z_{\alpha/2} = 1.960$) that the true mean μ lies within E=0.50 of the sample mean $\overline{\mu}_k$. Inserting these quantities into (3.13), we find that

$$k = (1.960 \cdot \frac{1.770}{0.50})^2 \approx 48 \qquad (3.14)$$

It is conjectured that (3.13) can also be used to estimate sample size when p(T) differs significantly from the normal, provided an accurate value can be established for the variance of p(T). Assuming no information about the dispersion of p(T) is available, one can obtain an estimate for σ^2 by choosing a small sample size (e.g., k=30) and using (3.6) to obtain an initial estimate for the population variance. Then, increase the sample size in small increments and monitor the convergence of the result produced by (3.6).

3.2.5.1.1.4. A Step-By-Step Procedure

In summary, the proposed statistical approach to estimating measures such as MA4 can be carried out as follows:

1. Select a sample size k that produces a sample mean $\overline{\mu}_k$ that is a good estimate of the true mean μ. Equation (3.13) may be helpful when choosing the appropriate value for k. Sample size will typically range from k=30 to k=1000.

2. Determine all localized fault classes for the given circuit. For example, a stuck-at-0 on any NAND gate input is equivalent to the NAND gate output stuck-at-1. The number N_1 of localized fault classes establishes an upper bound on N (recall that N is the number of detectable fault equivalence classes for the given circuit).

3. Select a representative fault from each fault class to establish the set $\{f_1, f_2, ..., f_{N_1}\}$.

4. Randomly select a fault from the set $\{f_1, f_2, ..., f_{N_1}\}$ with replacement.

5. Using a chosen test generation algorithm (PODEM, FAN, etc.), compute a test vector for the fault chosen in step (4). Record the CPU time required to compute the test vector. In this regard, a limit is typically placed on the number of backtracks that are allowed when attempting to generate a test for a chosen fault; this backtrack limit reduces overall test generation time since it constrains the time spent attempting to generate tests for redundant faults. Assuming we are performing an experiment to estimate MA4, the "number of CPU seconds per test vector for combinational logic," the CPU time that does not result in a test vector should be excluded when creating the set $\{T_1, T_2, ..., T_k\}$. In effect, we are assuming that all hard-to-detect faults that force the algorithm to exceed the backtrack limit are redundant, even though test vectors generated for other faults might possibly detect such hard-to-detect faults. (Alternatively, each backtrack limit CPU time should be included in the set $\{T_1, T_2, ..., T_k\}$ when the experiment is performed to generate an estimate for MA3, the "number of CPU seconds per fault for combinational logic"; the resulting confidence interval for MA3 can then be multiplied by N_1 to obtain a confidence interval for an upper bound on total CPU time required for test generation.) Return to step (4) when considering MA4 and the backtrack limit is exceeded.

6. Repeat steps (4) and (5) as necessary to obtain the set of CPU times $\{T_1, T_2, ..., T_k\}$.

7. Use equation (3.5) to compute $\overline{\mu}_k$, an estimate of μ.

8. Use equation (3.7) to compute $S_{\overline{\mu}}$, an estimate for the standard deviation of the approximately normally distributed sample mean. Use N_1, as found in step (2), in place of N to obtain a conservative estimate for $S_{\overline{\mu}}$.

9. Choose a confidence level C_L. For example, let $C_L = 0.98$.

10. Compute α, where $\alpha = 1 - C_L$. Letting $C_L = 0.98$, we observe that $\alpha = (1 - 0.98) = 0.02$.

11. From a table of values for the Student t-distribution, determine the value for $t_{\frac{\alpha}{2}}$ that corresponds to k-1 degrees of freedom and the known value of α. For example, k-1=120 and $\alpha = 0.02$ results in $t_{\frac{\alpha}{2}} = 2.358$.

12. Insert the known values for $\overline{\mu}_k, S_{\overline{\mu}}$, and $t_{\frac{\alpha}{2}}$ into (3.9) to obtain the desired confidence interval with chosen level of confidence C_L.

If the confidence interval is wider than desired for a chosen confidence level C_L, the interval can be reduced by increasing sample size k and repeating (or extending) the random experiment. In this regard, the increased value of k reduces both $S_{\overline{\mu}_k}$ and the quantity $t_{\frac{\alpha}{2}}$.

3.2.5.1.2. Using the Confidence Interval to Estimate Cost

The above step-by-step procedure for obtaining a confidence interval for a cost-related measure (e.g., measure MA4) relies on sampling theory to obtain the desired estimate. When developing the procedure, it was necessary to rely on the central-limit theorem to establish that the sample mean (viewed as a random variable) is normally distributed. For example, the random variable $\tilde{\mu}_k$, defined earlier as "the number of CPU seconds per test vector for combinational logic," is essentially normally distributed for sufficiently large sample size k.

Now suppose it is desired to determine the distribution of a new random variable Z_Q, where Z_Q is defined to be "the number of CPU seconds per Q test vectors for combinational logic." Observing that available information about the population mean μ is contained in the distribution of the sample mean $\tilde{\mu}_k$, Z_Q is assumed to be linearly related to $\tilde{\mu}_k$ as follows:

$$Z_Q = Q \cdot \tilde{\mu}_k \tag{3.15}$$

The probability density function for Z_Q is then a scaled version of that observed for $\tilde{\mu}_k$ with

$$E(Z_Q) = E(Q \cdot \tilde{\mu}_k) = Q \cdot \mu \tag{3.16}$$

and

$$Var(Z_Q) = Var(Q \cdot \tilde{\mu}_k) = Q^2 \cdot Var(\tilde{\mu}_k). \tag{3.17}$$

Observing (3.16) and (3.17), simple relationships exist between the expected value of and variance of Z_Q and that of $\tilde{\mu}_k$. Equation (3.9) can then be modified to obtain the following confidence interval for the mean of Z_Q, denoted by μ_Q as follows:

$$Q \cdot \overline{\mu}_k - t_{\frac{\alpha}{2}} \cdot Q \cdot S_{\overline{\mu}} < \mu_Q < Q \cdot \overline{\mu}_k + t_{\frac{\alpha}{2}} \cdot Q \cdot S_{\overline{\mu}}. \tag{3.18}$$

This relation is exemplified by considering an earlier example that produced equation (3.11), where the range on "number of CPU seconds per test vector for combinational logic," is $4.475 < \mu < 5.065$. For a known lower bound on test set size of Q=200 test vectors, the range on μ_{200}, the "number of CPU seconds per 200 test vectors for combinational logic" is found using equation (i3.18) as

$$200 \cdot 4.770 - 2.462 \cdot 200 \cdot 0.120 < \mu_{200} < 200 \cdot 4.770 + 2.462 \cdot 200 \cdot (0.120) \tag{3.19}$$

or

$$895 < \mu_{200} < 1013. \tag{3.20}$$

In the proposed step-by-step procedure for estimating MA4, the CPU time that does not result in a test vector is excluded from the set $\{T_1, T_2, ..., T_k\}$. Equation (3.20) then indicates that a lower bound on CPU time required to compute a set of test vectors for detecting all detectable single stuck-at faults in the hypothetical circuit lies in the interval 895 CPU seconds and 1013 CPU seconds (with 98% confidence).

The statistical approach to estimating test generation time is further demonstrated in reference [91], where a "test generation time" confidence interval is established for the 74181 ALU and the AM25S05 multiplier. In each case, the actual test generation time lies within the 95% confidence limits as expected.

3.2.5.2. Estimating Test Set Size

3.2.5.2.1. Bounds on Test Set Size

When evaluating a preliminary chip design that includes scan path testability, the question arises as to how many test vectors are required to detect all detectable faults in the various blocks of combinational logic. One approach to answering this question is to make use of an existing test generation algorithm to compute a test set for each block of combinational logic. This approach can be very expensive for large redundant combinational circuits as the computer time required to generate a test set increases exponentially with the number of circuit gates and primary inputs.

An alternative and relatively inexpensive approach to determining the testability of a combinational circuit is to establish bounds on test set size. Such bounds can be computed in worst-case polynomial time by using the test counting procedure proposed by Debany[83]. This procedure, which was developed by considering circuits that contain AONN (AND, OR, NAND, NOR) and XOR gates, is outlined and exemplified in the following paragraphs.

3.2.5.2.1.1. Test Counting Procedure

Results presented by Debany ([83], theorems 5.24 and 6.14) show that $O(G)$ operations are required to find the minimum or maximum cardinality of any complete test set for a given AONN or XOR fanout-free network containing G gates. In addition, the lower and upper bounds on test set size for a fanout-free network are not affected when they are fanout-free subnetworks of another network. These observations produced the following theorem for obtaining bounds on test set size for a general combinational logic network which may be irredundant or redundant[83].

Theorem 7.4: Let C be a general combinational network (irredundant or redundant), with set of irredundant test sets $\mathbf{I}'_C$ and set of irredundant test sequences $\mathbf{R}'_C$, and C is composed of n distinct maximal fanout-free subnetworks C_1, C_2, ... C_n. Then

a. $L_{max}\,(\mathbf{I}'_C) \leq \sum_{i=1}^{n} L_{max}\,(\mathbf{I}_{C_i})$.

b. $L_{max}\,(\mathbf{R}'_C) \leq \sum_{i=1}^{n} L_{max}\,(\mathbf{R}_{C_i})$.

c. Furthermore, if C is an irredundant network

$$L_{min}\,(\mathbf{I}_C) \geq \max_{i=1,....n} \{L_{min}(\mathbf{I}_{C_i})\} .$$

where,

$\mathbf{I}_C$, which exists if and only if C is irredundant, is the set composed of the following sets:

- All partial test sets that are irredundant and detect all single stuck-at-0 (stuck-at-1) faults and no single stuck-at-1 (stuck-at-0) faults.
- All complete test sets that are irredundant and detect all single stuck-at faults.

$\mathbf{I}_{C_i}$ is the set composed of all irredundant test sets for circuit C_i, where C_i is the standalone version of the ith fanout-free subnetwork of C,

$\mathbf{R}_{C_i}$ is the set composed of all irredundant test sequences for circuit C_i,

$L_{min}(\mathbf{I}_{C_i})$ is the minimal cardinality of any complete test set in $\mathbf{I}_{C_i}$,

$L_{max}(\mathbf{I}_{C_i})$ is the maximal cardinality of any complete test set in $\mathbf{I}_{C_i}$,

$L_{min}(\mathbf{R}_{C_i})$ is the minimal length of any complete test seq in $\mathbf{R}_{C_i}$, and

$L_{max}(\mathbf{R}_{C_i})$ is the maximal length of any complete test seq in $\mathbf{R}_{C_i}$.

This theorem and related discussion in [83] suggests the following procedure for establishing bounds on test set size for a given combinational logic network:

1. Identify all maximal fanout-free subnetworks in the given combinational logic network.

2. Establish bounds on the number of irredundant test vectors required to test each of the fanout-free subnetworks.

3. Use theorem 7.4 and the results from step (2) to establish bounds on test set size for the given combinational logic network.

3.2.5.2.1.2. Test Counting Experiments

Test counting experiments reported in [83] that use fanout-free subnetworks for estimating bounds on test set size are presented in Table 3.13. The 15 columns of Table 3.13 contain the following information:

- Column 1 gives the number of primary inputs to each network (N). Column 2 gives the number of gates in each model. This number is the count of primitives in the model and can vary according to modeling style. Column 3 gives the number of fanout-free subnetworks in each model.

- Every network was subjected to (single) fault simulation, where all 2^N input combinations (for networks with N primary inputs as given in Column 1) were graded for fault detection. Fault simulation was performed using the Hierarchical Integrated Test Simulator (HITS) in two modes. The first mode found the total number of faults undetected after 2^N tests. The second mode found the set of all faults detected by each of the 2^N tests. Column 4 gives the total number of single faults considered by HITS. Column 5 gives the number of undetectable faults as reported by HITS, that is, the number of faults that are not detected by any of the 2^N possible tests. Only the models with 0 undetectable faults are irredundant with respect to all single faults. Note that Columns 4 and 5 are not filled in for networks that contain XOR elements as primitive gates. This is because HITS, as with all known commercially available fault simulators, faults only the lines and so does not require an n-input XOR element to be tested with all 2^N input combinations. The greatest percentage of undetectable faults occurs in the case of Network 25(8.2%).

- Column 6 is the lower bound for $L_{min}(\mathbf{I}'_C)$, and Columns 7 and 8 are the upper bounds for $L_{max}(\mathbf{I}'_C)$ and $L_{min}(\mathbf{R}'_C)$, respectively, as given in Theorem 7.4. In giving a lower bound for $L_{min}(\mathbf{I}'_C)$ for each network, the assumption is made that $\mathbf{I}_C$ exists for each network (i.e., that all faults are detectable in each network) when in fact that is not true for 24 of the 39 networks. However, as noted above, "nearly all" of the faults are detectable in each network.

- In order to check the test count bounds, it would be desirable to find the true values for $L_{min}(\mathbf{I}'_C)$, $L_{max}(\mathbf{I}'_C)$, and $L_{max}(\mathbf{R}'_C)$. However, it is essentially impossible to find (and prove) the true values for these quantities for any of the networks considered here. In lieu of using algorithms to provide the exact answers are applied heuristics to find good estimates. One such heuristic is a greedy procedure which seeks to find a close-to-global-optimum result by finding a local optimum on each of a sequence of steps [94]. A greedy minimization procedure was applied to find the estimates for $L_{min}(\mathbf{I}'_C)$ given in Column 9, where the procedure sequentially selected tests that maximized the additional number of detected faults on each step. A greedy minimization procedure was also applied to find the estimates for $L_{max}(\mathbf{R}'_C)$ given in Column 10, where the procedure sequentially selected tests that minimized the additional number of detected faults on each step (although in order for the resulting collection of tests to be irredundant at least one new fault had to be detected by each test). Every number in Column 9 has been verified as being the cardinality of an irredundant test set. Every number in Column 10 has been verified as being the length of an irredundant test sequence, although each corresponding test set is irredundant. No similar computationally efficient means of estimating $L_{max}(\mathbf{I}'_C)$ is known.

- Columns 11-13 are derived using work published by Hayes[95]. For each network, Column 11 gives Π, the total number of distince input/output paths in the network. Column 12 gives Hayes' upper bound on $L_{min}(\mathbf{I}_C)$ based on the assumption that multidimensional path sensitization may be required (where N is the number of primary inputs and M is the number of primary outputs). Column 13 gives Hayes' upper bound based on the assumption that only single

path sensitization is required.

- One hundred test sequences were created by selecting tests at random until all detectable faults were detected. The test sequences obtained were then reduced to irredundant test sequences. Column 14 gives the mean length of each set of random test sequences. Column 15 gives the standard deviations of the lengths in Column 14.

The bounds of Hayes were included for comparison with results obtained by using fanout-free subnetworks. The Hayes bounds are true bounds but are extremely high when compared to the bounds for $L_{max}(\mathbf{I}'_C)$ and $L_{max}(\mathbf{R}'_C)$. The bounds for $L_{max}(\mathbf{I}'_C)$ and $L_{max}(\mathbf{R}'_C)$ are somewhat high because the results were obtained without considering information about the fanout structure of the general networks. Hayes' bounds are very high because he takes into account every possible consequence of reconvergent fanout.

The randomly generated irredundant test sequences are provided for comparison because many commercial test pattern generation systems perform in exactly that manner. Fault simulation is now far cheaper than deterministic test generation because of the availability of fast concurrent fault simulators and new special purpose simulation hardware that embed concurrent fault simulation algorithms.

The Reed-Muller canonical (RMC) forms for the network, Networks 38 and 39, are included not only because they each contain a large XOR fanout-free subnetwork but also out of interest in the use of RMC networks for enhancing testability. It was shown by Reddy[96] that for any N-variable RMC realization (including an extra control input) at most 3N+4 tests are required. Networks 38 and 39 realize 5-variable functions (even though they have 6 primary inputs) so the number of tests required is at most 19. This number is 1.73 times the greedy minimum and is greater than the mean number of tests in random irredundant test sequences. Tests were generated for the RMC form using the test generation algorithm of [97] to obtain a test sequence of length 19. The test sequence, applied in the order generated by the procedure, contained 12 irredundant tests. This is only one more test than the greedy minimum.

Table 3.13. Results of Test Counting Experiments

CKT No.	Microcircuit (Combinational)	COLUMN 1. Primary Inputs	2. Gates	3. Fanout-Free Networks	4. HITS Faults	5. Undetectable HITS Faults
1.	28LS517 ALU (all primary outputs) (NAND)	12	145	44	542	0
2.	(primary output 1: F3)	12	97	24	367	27
3.	(primary output 2: F2)	10	77	21	286	18
4.	(primary output 3: F1)	8	57	18	206	9
5.	(primary output 4: F0)	6	35	13	121	0
6.	(primary output 5: CN+4)	12	101	21	381	30
7.	(primary output 6: OVR)	12	115	29	438	30
8.	28LS517 ALU (all primary outputs) (NAND, XOR)	12	130	39	-	-
9.	(primary output 1: F3)	12	94	23	-	-
10.	(primary output 2: F2)	10	74	20	-	-
11.	(primary output 3: F1)	8	54	17	-	-
12.	(primary output 4: F0)	6	32	12	-	-
13.	(primary output 5: CN+4)	12	101	21	-	-
14.	(primary output 6: OVR)	12	112	28	-	-
15.	54LS85 MAGNITUDE COMPARATOR (all primary outputs)	11	31	11	159	0
16.	(primary output 1: A>B)	10	23	9	112	4
17.	(primary output 2: A=B)	9	17	5	76	0
18.	(primary output 3: A<B)	10	23	9	112	4
19.	54LS181 ALU (all primary outputs) (NAND)	14	112	38	387	12
20.	(primary output 1: G)	12	49	14	164	0
21.	(primary output 2: CN+4)	13	56	16	191	0
22.	(primary output 3: P)	10	27	11	98	0
23.	(primary output 4: F3)	14	65	20	219	3
24.	(primary output 5: F2)	12	53	17	178	3
25.	(primary output 6: A=B)	14	104	31	342	28
26.	(primary output 7: F1)	10	41	14	134	3
27.	(primary output 8: F0)	8	28	7	87	3
28.	54LS181 ALU (all primary outputs) (NAND, XOR)	14	80	34	-	-
29.	(primary output 1: G)	12	49	14	-	-
30.	(primary output 2: CN+4)	13	56	16	-	-
31.	(primary output 3: P)	10	27	11	-	-
32.	(primary output 4: F3)	14	57	19	-	-
33.	(primary output 5: F2)	12	45	16	-	-
34.	(primary output 6: A=B)	14	72	31	-	-
35.	(primary output 7: F1)	10	33	13	-	-
36.	(primary output 8: F0)	8	20	6	-	-
37.	SAMPLE NETWORK	5	6	2	24	0
38.	SAMPLE NETWORK RMC FORM (NAND)	6	52	20	167	0
39.	SAMPLE NETWORK RMC FORM (NAND, XOR)	6	20	13	-	-

Table 3.13: Results of Test Counting Experiments (Continued)

CKT No.	Microcircuit (Combinational)	COLUMN 6. L (l) min C (l.b.)	7. L (l') max C (u.b.)	8. L (R') max C (u.b.)	9. GREEDY MIN	10. GREEDY MAX
1.	25LS2517 ALU (all primary outputs) (NAND)	9	331	381	23	156
2.	(primary output 1: F3)	24	216	242	34	94
3.	(primary output 2: F2)	18	166	188	22	45
4.	(primary output 3: F1	12	117	135	22	54
5.	(primary output 4: F0)	11	65	78	16	33
6.	(primary output 5: CN+4)	29	221	246	41	123
7.	(primary output 6: OVR)	14	262	293	42	123
8.	25LS2517 ALU (all primary outputs) (NAND, XOR)	9	306	341	23	156
9.	(primary output 1: F3)	24	211	234	34	94
10.	(primary output 2: F2)	18	161	180	22	45
11.	(primary output 3: F1)	12	112	127	22	54
12.	(primary output 4: F0)	11	60	70	16	33
13.	(primary output 5: CN+4)	29	221	246	41	123
14.	(primary output 6: OVR)	14	257	285	42	123
15.	54LS85 MAGNITUDE COMPARATOR (all primary outputs)	11	120	137	24	58
16.	(primary output 1: A>B)	11	82	96	17	38
17.	(primary output 2: A=B)	11	53	63	11	15
18.	(primary output 3: A<B)	11	82	96	17	38
19.	54LS181 ALU (all primary outputs) (NAND)	9	206	251	15	109
20.	(primary output 1: G)	16	79	93	19	72
21.	(primary output 2: CN+4)	17	94	112	22	85
22.	(primary output 3: P)	11	32	58	12	44
23.	(primary output 4: F3)	15	108	130	19	86
24.	(primary output 5: F2)	12	85	103	15	64
25.	(primary output 6: A=B)	10	176	213	16	120
26.	(primary output 7: F1)	9	63	77	11	39
27.	(primary output 8: F0)	5	34	43	8	20
28.	54LS181 ALU (all primary outputs) (NAND, XOR)	9	174	211	15	109
29.	(primary output 1: G)	16	79	93	19	72
30.	(primary output 2: CN+4)	17	94	112	22	85
31.	(primary output 3: P)	11	52	58	12	44
32.	(primary output 4: F3)	15	100	120	19	86
33.	(primary output 5: F2)	12	77	93	15	64
34.	(primary output 6: A=B)	9	149	181	16	120
35.	(primary output 7: F1)	9	55	67	11	39
36.	(primary output 8: F0)	5	26	33	8	20
37.	SAMPLE NETWORK	6	10	14	6	12
38.	SAMPLE NETWORK RMC FORM (NAND)	5	86	109	11	30
39.	SAMPLE NETWORK RMC FORM (NAND, XOR)	5	52	61	11	30

Table 3.13: Results of Test Counting Experiments (Continued)

CKT No.	Microcircuit (Combinational)	COLUMN 11. Path Number Π	12. 3 Π -2N+M	13. 2 Π	14. Mean Length of 100 Random Irr. Sequences	15. Standard Deviation
1.	25LS2517 ALU (all primary outputs) (NAND)	4677	14013	9354	55.8	4.0
2.	(primary output 1: F3)	993	2956	1986	52.4	3.0
3.	(primary output 2: F2)	630	1871	1260	42.0	2.3
4.	(primary output 3: F1)	333	984	666	32.1	2.3
5.	(primary output 4: F0)	102	295	204	21.4	1.6
6.	(primary output 5: CN+4)	429	1264	858	66.3	3.5
7.	(primary output 6: OVR)	2190	6547	4380	74.3	3.6
8.	25LS2517 ALU (all primary outputs) (NAND, XOR)	–	–	–	–	–
9.	(primary output 1: F3)	–	–	–	–	–
10.	(primary output 2: F2)	–	–	–	–	–
11.	(primary output 3: F1)	–	–	–	–	–
12.	(primary output 4: F0)	–	–	–	–	–
13.	(primary output 5: CN+4)	–	–	–	–	–
14.	(primary output 6: OVR)	–	–	–	–	–
15.	54LS85 MAGNITUDE COMPARATOR (all primary outputs)	221	652	442	35.7	2.4
16.	(primary output 1: A>B)	98	275	196	26.5	1.9
17.	(primary output 2: A=B)	25	58	50	13.4	0.8
18.	(primary output 3: A<B)	98	275	196	26.5	2.0
19.	54LS181 ALU (all primary outputs) (NAND)	929	2773	1858	37.4	3.5
20.	(primary output 1: G)	56	145	112	36.9	2.9
21.	(primary output 2: CN+4)	81	218	162	43.4	3.4
22.	(primary output 3: P)	24	53	48	22.4	2.3
23.	(primary output 4: F3)	156	441	312	41.2	3.6
24.	(primary output 5: F2)	108	301	216	32.2	2.7
25.	(primary output 6: A=B)	384	1125	768	42.3	4.5
26.	(primary output 7: F1)	72	197	144	22.8	2.4
27.	(primary output 8: F0)	48	129	96	12.3	1.4
28.	54LS181 ALU (all primary outputs) (NAND, XOR)	–	–	–	–	–
29.	(primary output 1: G)	–	–	–	–	–
30.	(primary output 2: CN+4)	–	–	–	–	–
31.	(primary output 3: P)	–	–	–	–	–
32.	(primary output 4: F3)	–	–	–	–	–
33.	(primary output 5: F2)	–	–	–	–	–
34.	(primary output 6: A=B)	–	–	–	–	–
35.	(primary output 7: F1)	–	–	–	–	–
36.	(primary output 8: F0)	–	–	–	–	–
37.	SAMPLE NETWORK	5	6	10	7.6	1.1
38.	SAMPLE NETWORK RMC FORM (NAND)	1068	3193	2136	16.6	1.9
39.	SAMPLE NETWORK RMC FORM (NAND, XOR)	–	–	–	–	–

4. CHIP LEVEL DFT/BIT

The system developer must have information on performance and cost of alternative DFT/BIT techniques to select the appropriate chip level testability for a new chip design. To this end, task 2 of this program involves the assessment of DFT/BIT techniques for enhancing fault detection and diagnosis during the R&D, manufacture, and the operational phases. Test mode options such as on-line, off-line, built-in test, external test, and enhancements for fault-tolerance (e.g., correction, masking, and recovery), are also considered.

4.1. Alternative DFT/BIT Techniques

Numerous DFT/BIT techniques, and variations thereof, have been proposed for enhancing the testability of digital circuits at all levels of the system hierarchy. The focus of this research is at the chip level, and the techniques that are available for enhancing chip level testability can be categorized as follows:

- DFT techniques
  - Ad hoc
  - Structured
- Off-line/On-line BIT techniques
  - Off-line
  - On-line
- Fault-tolerant techniques
  - Hardware redundancy
  - Data redundancy
  - Time redundancy

These categories are further described in Figure 4.1, Figure 4.2, and Figure 4.3, where taxonomies DFT/BIT techniques are presented. Nearly all of the of DFT/BIT techniques presented in the taxonomies have been used at one time or another by the test engineers to enhance chip testability, and the system developer should consider using one or more of these techniques when selecting testability features for a new chip design.

Viewing the three DFT/BIT taxonomies, the question arises as to which of the many DFT/BIT techniques should be given high-priority by the system developer. That is, which of the techniques are known to be practical from the point of view of cost/performance? To answer this question, detailed descriptions of selected DFT, BIT, and fault-tolerant techniques, including performance and design penalty information, have been documented in a recent research report[2]. Results presented in the research report, combined with DFT/BIT research performed in this program, indicate that the sixteen DFT/BIT techniques have been used extensively by the test community and should be given primary consideration by the system developer. The sixteen techniques are identified and prioritized in the following section of this report.

4.2. DFT/BIT Selection

When specifying chip level DFT/BIT for a preliminary chip design, the testability objective is to select those techniques that ultimately satisfy system testability requirements in a cost-effective manner. This section discusses DFT/BIT cost/performance tradeoffs and identifies high-priority DFT/BIT techniques for consideration by the system developer as he strives to achieve this testability goal.

4.2.1. Cost/Performance tradeoffs

As discussed earlier in section 1.0, the system developer can use a combination of DFT/BIT to increase system reliability and maintainability as required to satisfy system requirements. More specifically, a combination of DFT (ad hoc and/or structured) and BIT (signature analysis, boundary scan, etc.) can be used to decrease the time required to detect and isolate a fault. This results in a reduction in MTTD

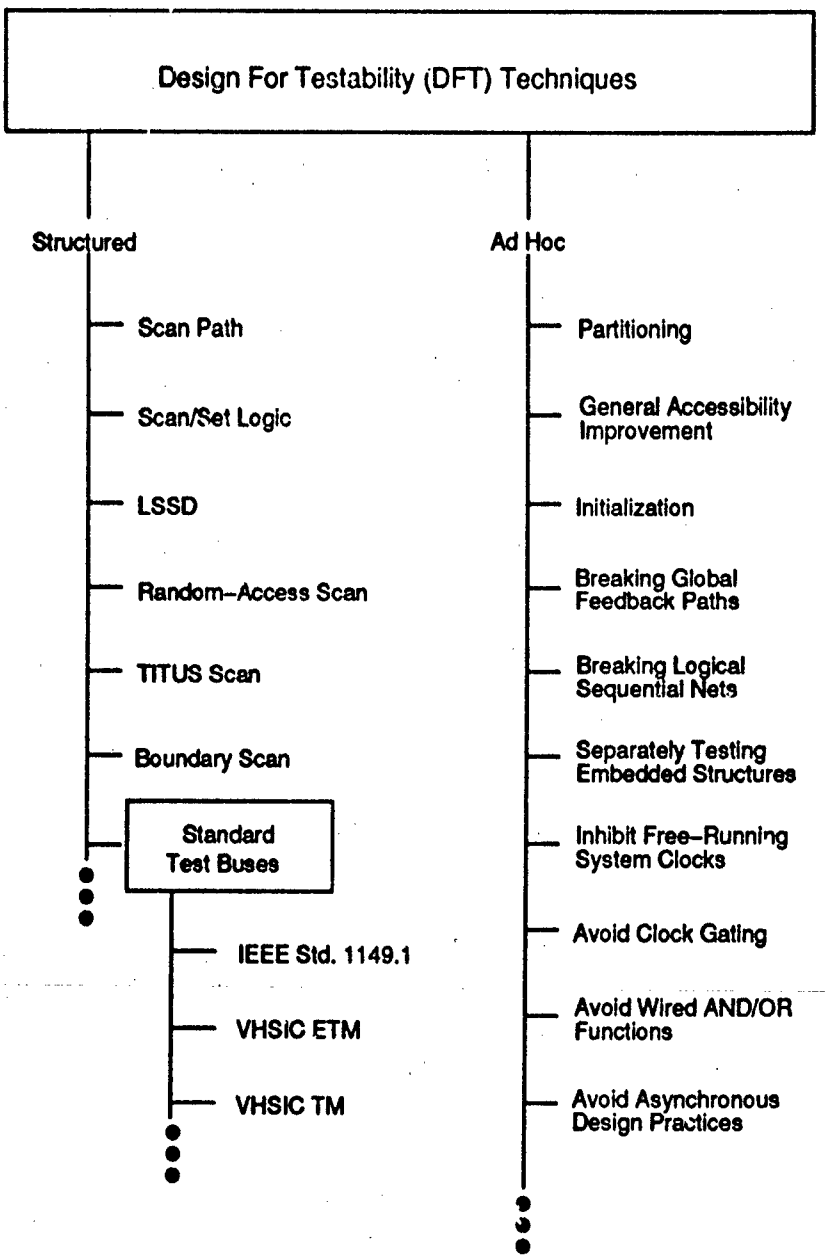

Figure 4.1. Taxonomy of DFT Techniques.

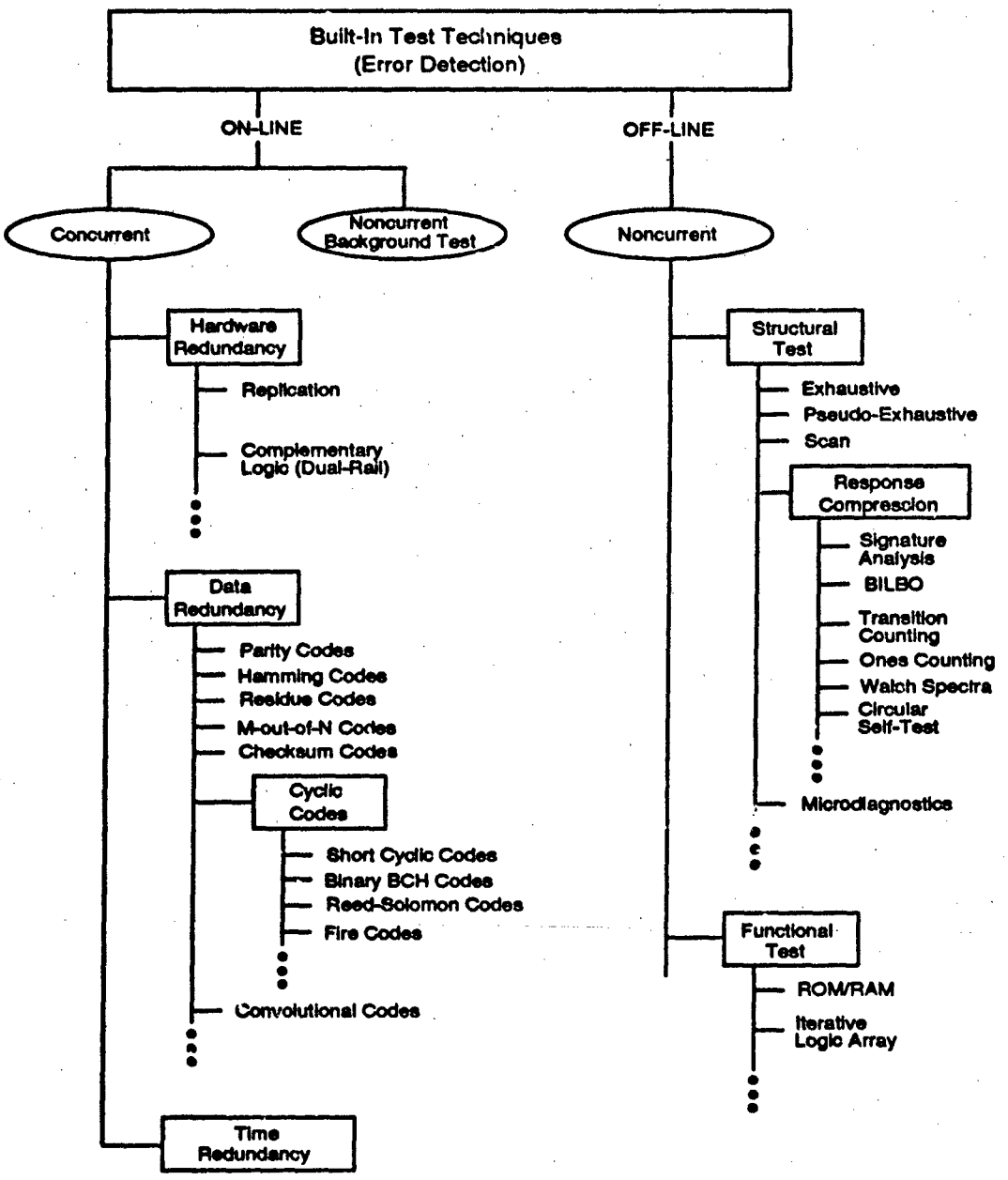

Figure 4.2. Taxonomy of BIT Techniques.

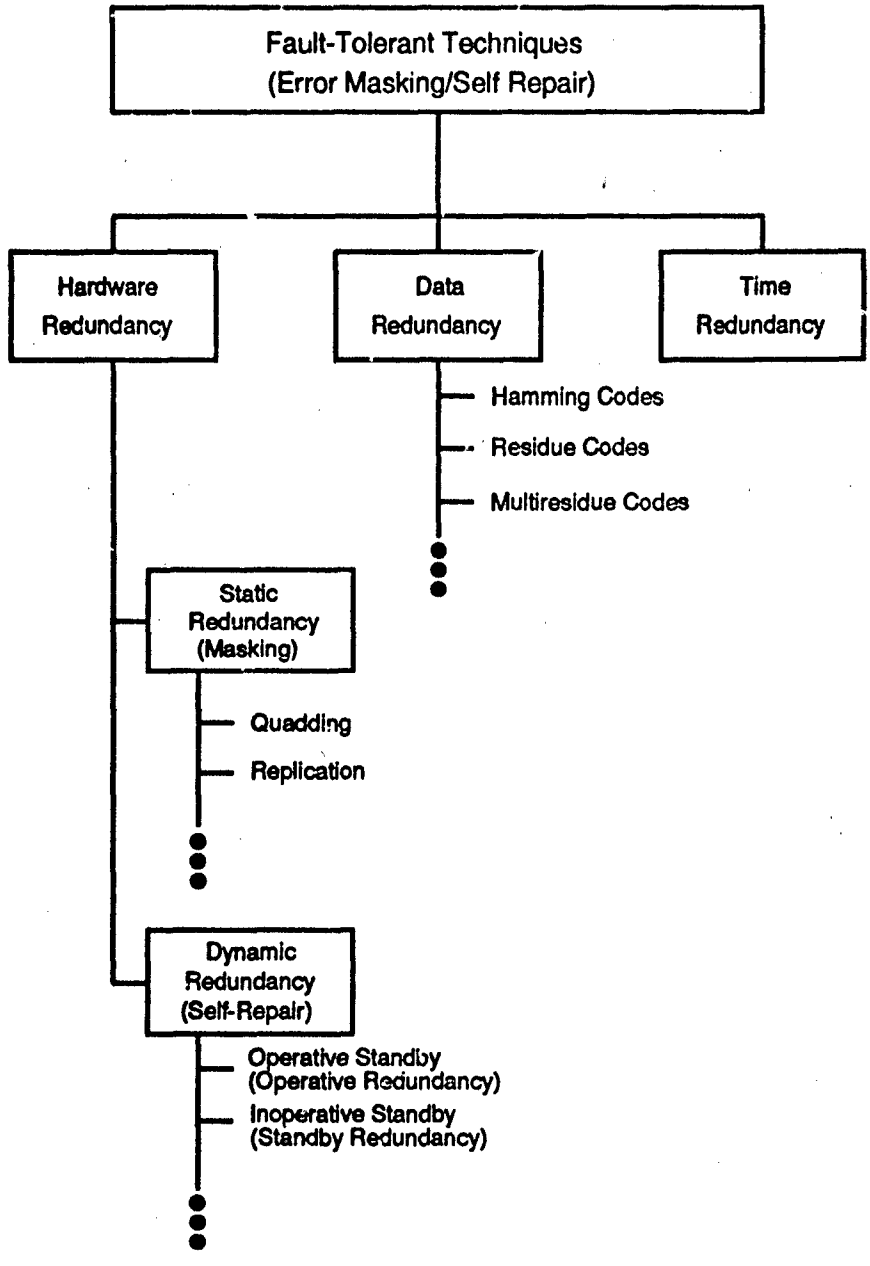

Figure 4.3. Taxonomy of Fault-Tolerant Techniques.

and MTTR which translates to an improvement in maintainability. On-line BIT (Hamming codes, replication, etc.) can also be used to mask errors which increases MTTF and improves reliability. A major component of the price to be paid for these improvements is the up-front research and development required to select the appropriate DFT/BIT for each chip design.

The quantities MTTD, MTTR, and MTTF are observed to be useful as system level performance metrics when evaluating the impact of alternative DFT/BIT techniques on system reliability and maintainability. These high level metrics can also be used at the chip level (they are functionally related to the testability factors and measures presented in section 3.0). In this regard, important chip level testability metrics for evaluating the effectiveness of DFT/BIT include the following:

1. *fault coverage* (fraction of detectable faults that are detected by a given test set and test application procedure)

2. *error latency* (time interval between first error occurrence and error detection, all errors being observed at the logic output)

3. *chip level MTTD* (the mean time required to detect the presence of a fault in a block of logic by a given test set and test application procedure)

The observation that chip level *Design for Testability* can increase fault coverage, decrease error latency, and decrease chip level MTTD *cannot be overemphasized.* The system developer must recognize that design for testability is absolutely essential for the development of testable VLSI chips. Also, design for testability at higher levels of the system hierarchy (use of test busses, boundary scan, etc.) is the only cost-effective way to satisfy stringent system testability requirements. For example, Williams and Parker[98] state that the number of test vectors N required to completely test a sequential machine from the I/O can range up to $N = 2^{m+n}$, where "m" is the number of machine states and "n" is the number of primary input lines[98]. A value of m=48 and n=16 results in $N = 2^{64}$ test vectors. Assuming 1 microsecond is required for application of each test vector, approximately 584,000 years would be required to apply 2^{64} test vectors to the circuit. This problem can be solved by using a *Design for*

Testability technique [2] to allow internal nodes to be controlled and observed with a relatively small number of test vectors. This example is included to emphasize that a carefully selected combination of DFT and BIT can be used to enhance reliability and maintainability in a cost-effective manner.

DFT/BIT design penalties to be considered at the chip level (and also at higher levels of the assembly) include the following:

- Design constraints imposed by a particular DFT/BIT technique
- Increase in development costs
- Increase in construction costs
- Performance degradation
- Silicon area cost
- Additional power required
- Number of additional I/O pins required
- Weight increase (at higher levels of assembly)
- Maintainability of BIT
- Reliability of BIT
- DFT/BIT software requirements

Available performance and design penalty information on several important DFT/BIT techniques appears in a recent research report[2] that focused on the assessment of DFT/BIT techniques. The techniques described in the research report are practical techniques that are generally recognized by the test community as being attractive techniques for consideration when developing testable and reliable digital systems.

4.2.2. High-Priority DFT/BIT Techniques

Prioritization of the DFT/BIT techniques shown in Figure 4.1, Figure 4.2, and Figure 4.3 is accomplished by considering the performance/cost of the techniques and their impact on system life-cycle cost. For example, in space-based applications the cost of loss of a satellite and the impact of such a loss on achieving mission objectives are especially important. Stated differently, the most important DFT/BIT techniques are those that increase system availability by increasing MTTF and decreasing MTTD and MTTR (see Section 1). Increased availability is achieved by using DFT/BIT techniques that are cost-effective and provide for thorough testing and fault-tolerance. Based on these criteria and knowledge of the various techniques as reflected in the data base information, sixteen of the most important DFT/BIT techniques are identified and prioritized immediately below.

High Priority DFT Techniques:

- Scan Design
  - Level-Sensitive Scan Design (LSSD) [2]
  - TITUS Scan [2]
- Boundary Scan [2]
- Standard Test Port (IEEE Std 1149.1) [99]

High Priority Off-line BIT Techniques:

- Scan Test (Off-line test performed on a scan design) [2]
- Signature Analysis [2]

High Priority On-line BIT Techniques:

- Parity Codes [2]
- Hamming Codes [2]

- M-out-of-N Codes [100,101]
- Checksum Codes [101,102]
- Residue Codes [101,103]
- Short Cyclic Codes [101,102]
- Binary BCH Codes [101,102]
- Reed-Solomon Codes [101,102]
- Fire Codes [101,102]
- Replication [2]

The high priority DFT and off-line BIT techniques listed above are state-of-the-art techniques for enhancing chip testability in a cost-effective manner. These techniques, which can also be used to detect and isolate faults at all levels of the system hierarchy, should be given primary consideration by the system developer when confronted with the problem of incorporating testability at the chip level.

Similarly, the high priority on-line BIT techniques are useful in applications that require detection and isolation of errors that occur while the system is performing its intended function. For example, the short cyclic codes refer to binary cyclic codes with relatively few check digits (other cyclic codes include the BCH codes, RS codes, and Fire codes). The short cyclic codes are used primarily for error detection or single-bit error correction. More specifically, a cyclic Hamming code with distance $d_{MH} = 3$ is a short cyclic code that can be used to detect single or double-bit errors that occur when data is transferred over a data path.

Recognizing that knowledge in the area of digital system testability is expanding rapidly due to the efforts of many researchers, the system designer should take note that the above list of practical DFT/BIT techniques is a "temporary" list that will expand with time. To this end, RTI has evaluated several homogeneous DFT/BIT techniques to identify promising techniques for consideration by the system developer. This research effort is described immediately below.

4.2.3. Evaluation of Homogeneous DFT/BIT Techniques

This section contains an evaluation of several homogeneous DFT/BIT techniques, where "homogeneous" refers to techniques that are suitable for incorporation into a structured design methodology. The chosen techniques are evaluated against a common set of criteria involving such factors as performance degradation, testability enhancement, overhead incurred, and suitability for automation in a CAD/CAE environment.

As circuit integration levels have increased rapidly over the years, so have the costs associated with testing these devices. To help alleviate these high costs caused in part by the reduced accessibility of internal nodes, a number of DFT and BIT techniques have been proposed either to improve the testability of integrated circuits or to make them self-testing. A few techniques such as scan design, and other scan path techniques, have been automated and incorporated into a CAD system which automatically implements the scan path into the design. However, scan path methods still may suffer from the exorbitant costs which are associated with automatic test pattern generation (ATPG).

An alternative method of reducing the costs associated with the testing of VLSI devices is through the ure of BIT techniques such as BILBO [104] and Signature Analysis (SA) [105] which provide an on-chip self-test capability. These techniques make use of a special structure called a Linear Feedback Shift Register (LFSR) [106] which is easily implemented in hardware. A maximal-length LFSR of length n can be used to generate $2^n - 1$ pseudorandom patterns which can be applied at-speed to the circuit-under-test (CUT). One advantage of using an LFSR for pseudorandom test pattern generation is that it allows one to test for delay faults (which may not necessarily be detected during deterministic testing) since the pseudorandom patterns can be applied at-speed. In addition, the time-consuming and costly task of generating and storing deterministic test patterns is eliminated.

A number of studies [107,108] have shown the economic benefits of incorporating self-test logic on VLSI circuits. The use of BIT techniques significantly impacts not only component testing, but has a positive impact on higher levels of testing

throughout the life-cycle. If BIT has as positive an impact as these studies show, the question arises as to why BIT has yet to be accepted in the design community? It seems a number of factors are involved which have, to date, prevented BIT from being widely used. One factor is the direct negative impact that BIT has on chip area and performance. Many designers are reluctant to the devote the 10–20% overhead that is typically required to implement BIT functions. Since most designers are not necessarily familiar with testability requirements, they typically do not regard testability as a design parameter.

An additional barrier to the acceptance of DFT and BIT methods is the lack of a structured design methodology for incorporating these techniques into the design and development phases. At present, most techniques that are incorporated into a design are implemented in an ad hoc fashion at the discretion of the design team. This is due to the lack of CAD tools that remove the burden of inserting DFT/BIT methods from the designers. Some progress has been made in recent years in the design automation of DFT and BIT features. CAD tools such as TITUS [109] and VENUS [110] have automated the incorporation of scan techniques into the design process. Although the incorporation of scan paths has made great inroads into reducing the complexity of testing VLSI devices, problems still exist with the high costs associated with test pattern generation, long test application times, and the need to store large amounts of data for these particular methods.

4.2.3.1. Alternative Techniques

RTI technical personnel searched technical journals, conference proceedings, books, and other available sources to identify alternative homogeneous DFT/BIT techniques. The literature search was conducted in two separate phases. In the initial literature search, a total of 16 techniques were identified as being potential techniques for homogeneous DFT/BIT. These techniques and a reference source are listed in Table 4.1. After conducting a feasibility study of each of these techniques, it was deemed that techniques 12, 13, 14, 15, and 16 were less promising than the others and were thus eliminated from further study. Additional literature references were then tracked down and studied to determine the technical capabilities and limitations of the re-

Table 4.1. Techniques Found During the Literature Search

Technique	Author	Reference
1) Circular BIST	Stroud	ITC 1989
2) Circular Self-Test Path (CSTP)	Pilarski	DAC 1987
3) Design Methodology Incorporating Self-Test (DEMIST)	Ambler	ITC 1986
4) Built-In Exhaustive Test (BEST)	Krasniewski	ITC 1985
5) Simultaneous Self-Test (SST)	Bardell	ITC 1982
6) LSSD On-Chip Self-Test (LOCST)	LeBlanc	Design & Test Nov. 1984
7) Pseudo-Exhaustive Test	McCluskey	Trans. on Computers June 1984
8) Weighted Random Test	Wunderlich	ITC 1988
9) ICL/ASTA	Illman	ITC 1989
10)LSSD	Eichelberger	DAC 1977
11) CrossCheck	Gheewala	DAC 1989
12) Cyclic Analysis Testing System	Burkness	U.S. Patent 4,680,761
13) External Self-Test Using Scan Path	Segers	ITC 1981
14) Mixed-Mode Self-Test	El-Ziq	ITC 1984
15) Centralized Verification Test	McCluskey	Custom IC Conf. 1987
16) L3-BILBO	Ohletz	ITC 1987

maining 11 techniques.

4.2.3.2. Evaluation Criteria

To make a proper evaluation and comparison of the DFT/BIT techniques identified above, RTI has developed a set of evaluation criteria to characterize and assess the capabilities and deficiencies of each technique. The criteria were developed by focusing on three distinctive areas: (1) performance, (2) cost, and (3) applicability.

The developed set contains nine distinctive criterion categories of which three

categories are divided into two subcategories each. Four of the criterion categories are used to evaluate the cost of each technique. Three categories deal with the performance of the techniques while the remaining two criterion categories are used to measure the applicability of the techniques. The criteria set is defined below.

Criterion #1 — Hardware Overhead

1a) Logic Overhead — The amount of extra logic needed to incorporate the chosen BIT technique in the design. The overhead will be defined in terms of additional 2-input CMOS gate equivalents. Low overhead will merit a high score.

1b) I/O Pin Overhead — the number of additional I/O pins needed to implement this technique. This has a large impact on packaging costs. A low number of extra I/O merits a high score.

Criterion #2 — Fault Coverage

2a) Stuck-at Coverage — The ratio of the number of faults detected by the technique to the number of detectable faults. This is a measure of the effectiveness of the BIT technique. The fault model assumed is the single stuck-at fault model. High coverages merit a high score.

2b) Delay/Opens/Bridging Faults Coverage — The applicability of the method for testing delay faults is the major concern of this criterion. Other types of faults, such as bridging faults, opens, shorts, etc., should also be given consideration. Good coverage of these types of faults merits a high score.

Criterion #3 — Test Run-Time

The number of clock cycles needed to run the entire BIT sequence. Some techniques may require more than a single test session to complete the test or may require some serial data shifting to set up and conclude the testing session. This is a measure

of the overall test application time. Also, some techniques require a slow clock for performing certain tasks which can significantly lengthen ue test time. Short test times merit high scores.

Criterion #4 — Performance Degradation

The reduction in chip performance (during the normal system mode of operation) caused by the delay associated with the additional BIT circuitry. Low degradation merits a high score.

Criterion #5 — Relative Ease of Automation

Although the techniques chosen for inclusion in this comparative investigation are all suitable for automation in a CAD environment, this criterion will measure the relative differences in the ability to automate different techniques. Techniques that are simple overlays of an existing design or that can be easily automated merit high scores.

Criterion #6 — External Accessibility and Interface Requirements

6a) External Accessibility by System Diagnostics/ATE — The ease with which maintenance processors or external ATE can access the results of the self-test sequence. In some cases, external ATE cannot control or monitor the testing procedure at the operational speed of the device under test. This may result in special needs for the test equipment to be compatible with a given BIT technique. Techniques with good accessibility merit high scores.

6b) JTAG Compatibility — Compatibility of the technique with the JTAG/IEEE 1149.1 Test Port standard. The test port can be used to support the self-test functions, transfer test data to and from the device under test, and pass self-tests status information to the maintenance processor. A technique which can use the JTAG standard to communicate control and test data to and from the device merits a high

score.

Criterion #7 — Applicability/Compatibility to Different Circuit Structures

The ability of the technique to be applied to different types of circuits such as datapath-intensive circuits, finite-state machine controllers, or circuits with various special logic structures such as RAM and PLA. In addition, this criterion will also reflect the compatibility of some techniques towards other structured BIT techniques such as Scan or special structured techniques for RAM, PLA, etc. Techniques which are applicable to a wide range of different circuit structures merit a high score.

Criterion #8 — Life-Cycle Usefulness

The ability of the chosen technique to be used throughout the life-cycle of the circuit at each level of testing that is required. Some techniques may not be useful during production testing due to the need for large amounts of serial data. In other cases, certain techniques may not be useful during field-level testing due to complex ATE interface requirements. This criterion will attempt to measure the degree to which the technique can be applied at all levels of testing. Techniques which can be used at many levels of testing throughout the life-cycle of a product merit a high score.

Criterion #9 — Expected Computational Expense

The amount of CPU time expected for the automation software to complete the translation of a design into a self-testing design. This criterion includes the computational expense that may be required to generate deterministic test vectors and/or the cost in computing the near-optimal weights needed for weighted random vectors. This criterion is important from the end-user's point of view in that he would like to know approximately how much CPU effort will be needed to do the design translation. Some consideration should also be given to the memory requirements (i.e., a few Mbytes versus a few hundred Mbytes). Techniques with low computational expenses

merit a high score.

4.2.3.3. Criteria Weighting Factors

To reflect the relative importance of the criteria during the preliminary screening process, a set of weighting factors was developed. These weighting factors are used to multiply the score assigned to each technique for the various criteria such that an overall figure of merit can be obtained which reflects the importance of each criterion. This figure of merit is obtained by summing the weighted criterion scores together.

During the life-cycle of an integrated circuit, the relative importance of these criteria may change. For example, maintenance technicians who service fielded equipment do not really care how difficult it was to automate the self-test technique in a circuit. Rather, they are more concerned with how quickly they can diagnose the failure and how difficult it will be to interface with the failed system. To reflect the changes in importance that can occur, three sets of weighting factors were developed. These weights reflect the relative importance of the criteria at the following phases: (1) the Research, Design, and Development (R&D) phase, (2) the Manufacturing phase, and (3) the phase where the equipment is operating in the field (the Field phase). Thus, a figure of merit score can be obtained for each technique from these three points of view. An overall figure of merit score can then be obtained by summing the individual scores. The weight factors are shown in Table 4.2.

4.2.3.4. Promising BIT Techniques

Each of the DFT/BIT techniques identified above was subjected to an evaluation using the twelve criteria developed previously. The evaluation was performed based on the technical information gathered during the literature search. For each criterion, a score of Low, Medium, or High was assessed for each technique based on its relative performance when compared to the other techniques being studied. Qualitative scores of this nature were used rather than a numerical score. An accurate numerical score for each criterion was difficult to assign based on the technical information that had been gathered. Even though some of the techniques had specific information regarding

Table 4.2. Criteria Weighting Factors

Criterion	R&D	Manufacturing	Field
Logic Overhead	15	15	0
I/O Overhead	5	10	5
Fault Coverage	15	15	15
Delay Fault Coverage	5	10	15
Test Run-Time	5	15	20
Performance Degradation	10	10	5
Ease of Automation	10	0	0
External Accessibility	5	10	15
JTAG Compatibility	5	5	15
Applicability to Ckts.	10	0	0
Life-Cycle Usefulness	5	10	10
Computational Expense	10	0	0
Total	100	100	100

their performance and costs, RTI personnel decided that only relative scores could be assigned at this point in the program. The scores assigned to each technique are shown in Table 4.3.

To use the weighting factors that were developed, these qualitative scores must be converted into a numerical score. To accomplish this, the following numerical scores are assigned:

Low	0 points
Medium	5 points
High	10 points

For each technique, the individual criterion scores are multiplied by the respective weighting factor for the R&D, Manufacturing, and Field points of view. The weighted criterion scores are then totaled to produce a total combined score for each of the three viewpoints as shown in Table 4.4. These totals are then further summed together to produce an overall figure of merit as shown in the last column.

Based on the results shown in Table 4.4, it was observed that three techniques scored significantly above the average score of 194.5. These three techniques were (in order) (1) CrossCheck, (2) the LSSD On-Chip Self-Test (LOCST) technique, and (3) the Circular BIST technique. A detailed description of each of these potentially attractive DFT/BIT techniques appears in Appendix A of this report.

Table 4.3. Relative Scores for Each DFT/BIT Technique

Criterion	Technique					
	Circular BIST	CSTP	LOCST	BEST	DEMIST	SST
Logic Overhead	M	M	H	L	L	L
I/O Pin Overhead	H	H	H	M	M	H
Fault Coverage	M	M	H	H	H	M
Delay Fault Coverage	H	H	M	H	H	M
Test Run-Time	H	H	L	M	M	H
Performance Degradation	M	L	M	M	L	M
Ease of Automation	L	L	H	L	L	H
External Accessibility	H	M	H	H	H	M
JTAG Compatibility	H	M	H	H	H	H
Applicability to Different Ckts.	M	M	H	L	H	H
Life-Cycle Usefulness	M	L	M	M	M	M
Computational Expense	L	L	H	L	L	H

Legend:
L = Low
M = Medium
H = High

Table 4.3. Relative Scores for Each DFT/BIT Technique (continued)

Criterion	Technique				
	Pseudo-exhaustive Test	Weighted Random Test	ICL/ ASTA	LSSD	CrossCheck
Logic Overhead	M	M	L	H	H
I/O Pin Overhead	M	M	L	M	M
Fault Coverage	H	H	H	H	H
Delay Fault Coverage	H	H	H	M	M
Test Run-Time	M	M	M	M	M
Performance Degradation	M	M	M	M	H
Ease of Automation	M	H	L	H	M
External Accessibility	L	L	H	L	H
JTAG Compatibility	H	H	M	H	H
Applicability to Different Ckts.	M	M	H	H	H
Life-Cycle Usefulness	M	M	M	M	H
Computational Expense	M	M	M	L	M

Table 4.4. Preliminary Screening Scores

Technique	R&D	Manufacturing	Field	Total
Circular BIST	52.5	75	85	212.5
CSTP	40	57.5	62.5	160
LOCST	85	70	65	220
BEST	42.5	62.5	80	185
DEMIST	47.5	57.5	77.5	182.5
SST	65	57.5	70	192.5
Pseudo-exhaustive Test	60	60	65	185
Weighted Random Test	65	60	65	190
ICL/ASTA	52.5	55	70	177.5
LSSD	70	62.5	57.5	190
CrossCheck	82.5	82.5	80	245

5. CONCLUSIONS

The objective of this research effort is to establish guidelines that will assist system developers in specifying consistent, necessary, and achievable chip level testability requirements. Such guidelines will contribute to solving the more general problem of setting cost-effective testability requirements at all levels of the system hierarchy.

Section 2 of this report discusses two approaches (top-down and bottom-up strategies) to setting testability requirements. The top-down approach consists of distributing system testability requirements to lower levels and ultimately to the chip level. The top-down approach is not attractive since the resulting chip requirements may be unnecessary or unachievable. Such requirements ultimately result in chip testability enhancement that is not cost-effective.

Alternatively, the bottom-up approach is a procedure for setting chip testability requirements that are consistent, necessary, and achievable. Testability evaluation of the preliminary chip design, containing judiciously chosen DFT/BIT, is an essential step in bottom-up approach to specifying optimum (or near optimum) testability requirements. Testability is evaluated by estimating testability attributes of the proposed chip design. The resulting DFT/BIT cost/performance tradeoffs then provide a basis for establishing cost-effective testability requirements. A set of recommended testability measures, along with methods for estimating the measures with meaningful bounds or confidence intervals for the estimates, appear in Section 3.

Section 4 presents a comprehensive list of testability techniques and identifies sixteen high-priority DFT/BIT techniques that should be given primary consideration when incorporating testability into a new chip design. These selective use of these chip level testability techniques can result in cost-effective testability enhancement of new chip designs. Several homogeneous DFT/BIT techniques are also evaluated to identify promising techniques for consideration by the system developer.

References

[1] J. W. Watterson, M. Royals, N. Vasanthavada, J. J. Hallenbeck, and N. Kanopculos. Testability Measurement and BIT Evaluation (Volume I: Testability Measurement). *Rome Laboratory Report*, In Publication.

[2] J. W. Watterson, M. Royals, J. J. Hallenbeck, and N. Kanopoulos. Testability Measurement and BIT Evaluation (Volume II: BIT Evaluation). *Rome Laboratory Report*, In Publication.

[3] J. W. Watterson, M. Royals, N. Vasanthavada, J. J. Hallenbeck, and N. Kanopoulos. Testability Measurement and BIT Evaluation (Volume III: Appendices). *Rome Laboratory Report*, In Publication.

[4] J. P. Hayes. Fault Modeling. *IEEE Design and Test*, pages 88–95, 1985.

[5] W. H. Debany. Digital Logic Testing and Testability. *Rome Laboratory In-House Report No. RL-TR-91-6*, 1991.

[6] D. P. Siewiorek, V. Kini, H. Mashburn, S. R. McConnel, and M. M. Tsao. A case study of C.mmp, Cm*, and C.vmp: Part I – Experiences with Fault Tolerence in Multiprocessor Systems. *Proceedings of the IEEE*, pages 1178–1199, October 1978.

[7] J. W. Watterson, J. J. Hallenbeck, G. A. Frank, D. L. Franke, and J. B. Clary. Tools and Techniques for Assessment of VHSIC On-Chip Self-Test and Self-Repair. *RTI/2086/01-1F, Research Triangle Institute*, February 1985.

[8] M. Schulz and E. Trischler. Applications of Testability Analysis to ATG: Methods and Experimental Results. *Proc. Int. Conf. on Circuits and Systems*, pages 691–694, 1985.

[9] J. P. Roth. Diagnosis of Automata Failures: A Calculus and a Method. *IBM J. Res. Develop*, pages 278–291, 1966.

[10] P. Goel. An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits. *IEEE Trans. Comp.*, pages 215–222, 1981.

[11] H. Fujiwara and T. Shimono. On the Acceleration of Test Generation Algorithms. *IEEE Trans. Comp.*, pages 1137–1144, 1983.

[12] E. Trischler. Guided Inconsistent Path Sensitization: Methods and Experimental Results. *Proc. IEEE Int. Test Conf.*, pages 79–86, 1985.

[13] H. Y. Chang, E. Manning, and G. Metze. *Fault Diagnosis of Digital Systems.* John Wiley & Sons, 1970.

[14] J. P. Roth, W. G. Bouricius, and P. R. Schneider. Programmed Algorithms to Compute Tests to Detect and Distinguish Between Failures in Logic Circuits. *IEEE Transactions on Electronic Computers*, pages 567–580, 1967.

[15] F. J. Hill and B. Huey. SCIRTSS: A Search System for Sequential Circuit Test Sequences. *IEEE Transactions on Computers*, pages 490–502, 1977.

[16] M. A. Breuer and A. D. Friedman. TEST/80 A Proposal for an Advanced Automatic Test Generation System. *Proceedings of the IEEE AUTOTESTCON*, pages 305–312, 1979.

[17] P. Goel and B. C. Rosales. PODEM-X: An Automatic Test Generation System for VLSI Logic Structures. *18th Design Automation Conference*, pages 260–267, 1981.

[18] H. Fujiwara. *Logic Testing and Design for Testability.* MIT Press, Cambridge, Massachusetts, 1985.

[19] *HITS User Guide*, Lakehurst, NJ, 1987. Naval Air Engineering Center.

[20] Editor. Guide to Fault Simulators. *VLSI Design*, pages 31–33, 1984.

[21] E. Trischler. ATWIG, An Automatic Test Pattern Generator with Inherent Guidance. *Proceedings of the IEEE International Test Conference*, pages 80–87, 1984.

[22] M. J. Bending. HITEST: A Knowledge-based Test Generation System. *IEEE Design & Test*, pages 83–92, 1984.

[23] S. Shteingart, A. W. Nagle, and J. Grason. RTG: Automatic Register Level Test Generator. *Proceedings of the 22nd Design Automation Conference*, pages 803–807, 1985.

[24] M. Abramovici, J. J. Kulikowski, P. R. Menon, and D. T. Miller. SMART and FAST: Test Generation for VLSI Scan-Design Circuits. *IEEE Design and Test*, pages 43–54, 1986.

[25] R. Lisanke, F. Brglez, A. deGeus, and D. Gregory. Testability-driven Random Pattern Generation. *Proceedings of the International Conference on Compu er-aided Design (ICCAD)*, pages 144–147, 1986.

[26] M. H. Schulz, E. Trischler, and T. M. Sarfert. SOCRATES: A Highly Efficient Automatic Test Pattern Generation System. *Proceedings of the IEEE International Test Conference*, pages 1016–1026, 1987.

[27] C. W. Cha, W. E. Donath, and F. Ozguner. 9-V Algorithm for Test Pattern Generation of Combinational Digital Circuits. *IEEE Transactions on Computers*, pages 193–200, 1978.

[28] Y. Takamatsu, M. Hiraishi, and K. Kinoshita. Test Generation for Combinational Circuits by a Ten-valued Calculus. *Transactions of Information Processing Society of Japan (In Japanese)*, pages 542–548, 1983.

[29] S. B. Akers. A Logic System for Fault Test Generation. *IEEE Transactions on Computers*, pages 37–42, 1976.

[30] D. B. Day and K. W. Warren. The Integration of Design and Test. *VLSI Design*, pages 46–52, 1985.

[31] P. Goel. Test Generation Costs Analysis and Projections. *Proc. 17th Design Automation Conference*, pages 77–84, 1980.

[32] Jonah McCleod. Gateway's New Simulator Turns up Faults in a Hurry. *Electronics*, pages 63–65, May 1988.

[33] T. Chan and E. Law. MegaFAULT: A Mixed-node, Hardware Accelerated Concurrent Fault Simulator. *Proceedings of the International Conference on Computer-aided Design*, pages 394–397, 1986.

[34] H. Shih, J. T. Rahmeh, and J. A. Abraham. FAUST: An MOS Fault Simulator with Timing Information. *IEEE Transactions on Computer-aided Design (ICCAD)*, pages 557–563, 1986.

[35] S. K. Jain and V. D. Agrawal. STAFAN: An Alternative to Fault Simulation. *Proc. 21st Design Automation Conference*, pages 18–23, 1984.

[36] S. K. Jain and V. D. Agrawal. Statistical Fault Analysis. *IEEE Design & Test of Computers*, pages 38–44, 1985.

[37] S. K. Jain and D. M. Singer. Characteristics of Statistical Fault Analysis. *Proc. Int. Conference on Computer Design*, pages 24–30, 1986.

[38] S. Miyamoto, T. Mishida, T. Kozawa, and F. Tada. TRUE: A Fast Detectability Estimation System. *Proceedings of the International Conference on Computer-aided Design (ICCAD)*, pages 36–37, 1983.

[39] T. Nishida, S. Miyamoto, and T. Kozawa. Performance Analysis of the Detectability Estimation System (TRUE). *Proceedings of the International Symposium on Circuits and Systems (ISCAS)*, pages 1317–1320, 1985.

[40] F. Brglez. A Fast Fault Grader: Analysis and Applications. *Proc. IEEE Int. Test Conf.*, pages 785–794, 1985.

[41] M. Kawamura and K. Hirabayashi. AFS: An Approximate Fault Simulator. *Proceedings of the IEEE International Test Conference*, pages 717–721, 1985.

[42] M. Abramovici, P. R. Menon, and D. T. Miller. Critical Path Tracing, An Alternative to Fault Simulation. *Proc. 20th Design Automation Conference*, pages 214–220, 1983.

[43] V. D. Agrawal and M. R. Mercer. Testability Measures – What Do They Tell Us? *Proc. IEEE Int. Test Conf.*, pages 391–396, 1982.

[44] S. Patel and J. Patel. Effectiveness of Heuristics Measures for Automatic Test Pattern Generation. *Proc. 23rd Design Automation Conf.*, pages 547–552, 1986.

[45] V. D. Agrawal, S. C. Seth, and C. C. Chuang. Probabilistically Guided Test Generation. *Proc. Int. Conf. on Circuits and Systems*, pages 687–690, 1985.

[46] E. Trischler. A Methodology for Statistical Evaluation of Estimated and Real Testability Measures. *Siemens Forsch. -u. Entwickl. -Ber Bd. 1?*, pages 1–8, 1987.

[47] R. Sedmak. private communication. *Self-Test Services*, 1986.

[48] W. M. Consolla and F. G. Danner. An Objective Printed Circuit Board Testability Design Guide and Rating System. *RADC TR-79-327*, 1980.

[49] J. C. Bussert. Printed-Circuit Board Testability Measurement Systems: Survey and Comparative Analysis. *NOSC Technical Document 743*, 1984.

[50] J. C. Bussert. Testability Measures on a State-of-the-Art Circuit. *NOSC Technical Document 835*, 1986.

[51] MIL-STD-2165. Testability Programs for Electronic Systems and Equipments, 1985.

[52] P. G. Kovijanic. Single Testability Figure of Merit. *Proceedings of the IEEE International Test Conference*, pages 521–528, 1981.

[53] J. Grason. TMEAS, A Testability Measurement Program.. *Proceedings of the 16th Design Automation Conference*, pages 156–161, 1979.

[54] L. H. Goldstein and E. L. Thigpen. SCOAP: SANDIA Controllability/Observability Analysis Program. *Proceedings of the 17th Design Automation Conference*, pages 190–196, 1980.

[55] L. H. Goldstein. Controllability/Observability Analysis of Digital Circuits. *IEEE Trans. on Circuits and Systems*, pages 685–693, 1979.

[56] B. Dunning and P. Kovijanic. Demonstration of a Figure of Merit for Inherent Testability. *Proceedings of the IEEE AUTOTESTCON*, pages 515–520, 1981.

[57] R. G. Bennets *et. al.* CAMELOT: A Computer-aided Measure for Logic Testability. *Proceedings of the IEEE*, pages 177–189, 1981.

[58] D. K. Goel and R. M. McDermott. An Interactive Testability Analysis Program-ITTAP. *Proceedings of the 19th Design Automation Conference*, pages 581–586, 1982.

[59] W. C. Berg and R. D. Hess. COMET: A Testability Analysis and Design Modification Package. *Proceedings of the IEEE International Test Conference*, pages 364–378, 1982.

[60] et al. I. M. Ratiu. VICTOR: A Fast VLSI Testability Analysis Program. *Proc. Int. Test Conf.*, pages 397–400, 1982.

[61] T. Kirkland and V. Flores. Software Checks Testability and Generates Tests of VLSI Design. *Electronics*, pages 120–124, 10, 1983.

[62] E. Trischler. An Integrated Design for Testability and Automatic Test Pattern Generation System: An Overview. *Proceedings of the 21st Design Automation Conference*, pages 209–215, 1984.

[63] Design for Repair Concepts and Applicable Tools. Project 412U-3860, Research Triangle Institute, 1987.

[64] *CAFIT User Manual*, California, 1986. ATAC Corporation.

[65] M. Kawai S. Takasaki and A. Yamada. A Calculus of Testability Measure at the Functional Level. *Proceedings of the IEEE International Test Conference*, pages 95–101, 1981.

[66] F. Brglez, P. Pownall, and R. Hum. Applications of Testability Analysis: from ATPG to Critical Delay Path Tracing. *Proc. IEEE International Test Conference*, pages 705–712, 1984.

[67] S. C. Seth, L. Pan, and V. D. Agrawal. FREDICT – Probabilistic Estimation of Digital Circuit Testability. *Proc. 15th Symp. Fault Tolerant Computing*, pages 220–225, 1985.

[68] H. J. Wunderlich. PROTEST: A Tool For Probabilistic Analysis. *Proc. 22nd Design Automation Conference*, pages 204–211, 1985.

[69] J. A. Dussault. A Testability Measure. *Proceedings of the Semiconductor Test Conference*, pages 113–116, 1978.

[70] R. G. Bennetts. *Design for Testable Logic Circuits.* Addison-Wesley, London, 1984.

[71] *The STAT Tutor.* DETEX Systems, Inc., October 1991.

[72] W. R. Simpson and H. S. Balaban. The ARINC Research System Testability and Maintenance Program (STAMP). *Proceedings of The IEEE AUTOTEST-CON*, pages 88–95, 1982.

[73] G. de Mare, P. J. Giordano, and M. Thiel. TRI-MOD... A Mission Effectiveness Testability Analysis Model. *Proceedings of the IEEE AUTOTESTCON*, pages 250–255, 1984.

[74] J. A. Abraham and W. A. Rogers. HAT: A Heuristic Advisor for Testability. *Proceedings of the International Test Conference on Computer Design*, pages 566–569, 1985.

[75] J. Byren, L. Deight, and G. Stratton. RADC Testability Notebook. *RADC-TR-82-189*, 1977.

[76] W. Keiner and R. West. Testability Measures. *Proceedings of The IEEE AUTOTESTCON*, pages 49–55, 1977.

[77] O. H. Ibarra and S. K. Sahni. Polynomially Complete Fault Detection Problems. *IEEE Transactions on Computers*, pages 242–249, 1975.

[78] J. W. Watterson and J. J. Hallenbeck. Modulo 3 Residue Code Checker: New Results on Performance and Cost. *IEEE Transactions on Computers*, 1988.

[79] J. Savir, G. S. Ditlow, and P. H. Bardell. Random Pattern Testability. *IEEE Transactions on Computers*, pages 79–90, 1984.

[80] B. Krishnamurthy and R. L. Sheng. A New Approach to the Use of Testability Analysis in Test Generation. *International Test Conference*, pages 796–778, 1985.

[81] S. B. Akers and B. Krishnamurthy. On the Application of Test Counting to VLSI Testing. *Chapel Hill Conference on VLSI*, pages 343–359, 1985.

[82] A. C. Hung and F. C. Wang. A Method for Test Generation Directly from Testability Analysis. *International Test Conference*, pages 62–78, 1985.

[83] W. H. Debany. *On Using Fanout-free Substructure of General Combinational Networks.* PhD thesis, Syracuse University, Dept. of Computer and Information Science, Syracuse, New York, 1985.

[84] *Testing of Random Access Memories, Theory and Practice*, The Netherlands, 1986. Eindhoven University of Technology.

[85] H. Fujiwara. Design of PLAs with Random Pattern Testability. *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems*, pages 5–10, 1988.

[86] D. T. Tang and C. L. Chin. Logic Test Pattern Generation Using Linear Codes. *Proc. Int. Symp. on Fault-Tolerant Computers*, pages 222–226, 1983.

[87] N. Vasanthavada and P. M. Marinos. An Operationally Efficient Scheme for Exhaustive Test Generation Using Linear Codes. *Proc. Int. Test Conf.*, pages 476–482, 1985.

[88] R. M. Sedmak. BIST Tutorial. *Int. Test Conf.*, 1987.

[89] E. J. McCluskey. Verification Testing, A Pseudoexhaustive Technique. *IEEE Transactions on Computers*, pages 541–546, 1984.

[90] V. D. Agrawal. An Information Theoretic Approach to Digital Fault Testing. *IEEE Transactions on Computers*, pages 582–587, 1981.

[91] J. W. Watterson, N. Vasanthavada, D. M. Royals, and N. Kanopoulos. Estimation of Cost-Related Testability Measures. *GOMAC-88 Digest of Papers*, pages 551–555, November 1988.

[92] A. M. Mood and F. A. Graybill. *Introduction to the Theory of Statistics.* McGraw Hill, 1963.

[93] K. S. Trivedi. *Probability & Statistics with Reliability, Queuing, and Computer Science Applications.* Prentice-Hall, 1982.

[94] et al. A. V. Aho. *Data Structures and Algorithms.* Addison-Wesley, 1983.

[95] J. P. Hayes. Path Complexity of Logic Networks. *IEEE Transactions on Computers*, pages 459–462, May 1978.

[96] S. M. Reddy. Easily Testable Realizations for Logic Functions. *IEEE Transactions on Computers*, pages 1183–1188, 1972.

[97] M. A. Breuer and A. D. Friedman. *Diagnosis and Reliable Design of Digital Systems.* Computer Science Press, Inc., 1976.

[98] T. W. Williams and K. P. Parker. Testing Logic Networks and Designing for Testabiiity. *IEEE Computer*, pages 9–21, October 1979.

[99] C. M. Maunder and R. E. Tulloss. *The Test Access Port and Boundary-Scan Architecture.* IEEE Computer Society Press, 1990.

[100] John Wakerly. *Error-Detecting Codes, Self-Checking Circuits and Applications.* North-Holland, 1978.

[101] D. P. Siewiorek and R. S. Swartz. *The Theory and Practice of Relilable System Design.* Digital Press, 1982.

[102] S. Lin and Jr. D. J. Costello. *Error Control Coding: Fundamentals and Applications.* Prentice-Hall, 1983.

[103] N. S. Szabo and R. I. Tanaka. *Residue Arithmetic and its Applications to Computer Technology.* McGraw-Hill, 1967.

[104] et al. B. Koenemann. Built-In Logic Block Observation Techniques. *Proc. IEEE Int. Test Conference*, pages 37 -- 41, 1979.

[105] R. A. Frohwerk. Signature Analysis: A New Digital Field Service Method. *Hewlett-Packard Journal*, pages 2 – 8, May 1977.

[106] L. T. Wang and E. J. McCluskey. Feedback Shift Registers for Self-Testing Circuits. *VLSI Systems Design*, pages 50 – 58, December 1986.

[107] P. Varma, A. B. Ambler, and K. Baker. An Analysis of the Economics of Self-Test. *Proc. IEEE Int. Test Conference*, pages 20 – 30, 1984.

[108] H. H. Butt and Y. M. El-ziq. Impact of Mixed-Mode Self-Test on Life Cycle Cost of VLSI Based Designs. *Proc. IEEE Int. Test Conference*, pages 338 – 347, 1984.

[109] V. D. Agrawal, S. K. Jain, and D. M. Singer. Automation in Design for Testability. *Proc. IEEE Custom Integrated Circuits Conference*, pages 159 – 163, 1984.

[110] H. G. Thonemann, M. Kolonko, and H. Severloh. VENUS—An Advanced VLSI Design Environment for Custom Integrated Circuits with Macro Cells, Standard Cells, and Gate Arrays. *Proc. IEEE Custom Integrated Circuits Conference*, pages 492 – 497, May 1987.

APPENDIX A

Technical Summaries of Three Homogeneous DFT/BIT Techniques

Contents

List of Figures

List of Tables

1. STRUCTURED BIT TECHNIQUES

FULL NAME OF TECHNIQUE:

Technique #1 - Circular Built-In Self-Test

DEFINITION OF TECHNIQUE:

Circular BIST is an off-line BIT technique which provides both pseudorandom test generation and test response compaction at the chip-level and is based on the feedback shift register approach. In this technique, system registers are selectively replaced by special BIST registers which are connected in a circular path forming a feedback shift register. The construction of this feedback shift register creates a data compaction capability with the results of the compactor used as the input stimuli in the next clock cycle for the circuitry under test. Upon completion of the self-test sequence, the chip status can be held in storage until it can be accessed by system diagnostics or ATE equipment.

DESCRIPTION OF TECHNIQUE:

In the Circular BIST technique, system registers are selectively replaced by a special BIST flip-flop which can operate in one of four modes; reset, shift, system, and BIST. Each BIST flip-flop contains two data inputs; a system data input and a serial data input. The BIST flip-flops are each connected in a chain by linking the serial input of one BIST flip-flop to the output of a previous BIST flip-flop. The last BIST flip-flop in the chain is then connected to the serial input of the first BIST flip-flop, thereby forming a circular path as shown below in Figure 1.1.

In addition to the chain of BIST flip-flops, a Signature Analysis Register (SAR) is included in the Circular BIST path in order to compact the output response of the Circular BIST chain into a reference signature. The SAR is included within the Circular BIST chain in order to increase its length and reduce the probability that the circular path will enter a short loop of repeating test patterns known as limit cycling.

During each clock cycle of the BIST sequence, the output response of the CUT is compacted into a signature which is applied as the next test pattern to the circuit. A BIST controller located on-chip initiates the BIST sequence by resetting all BIST and SAR flip-flops. By applying a single clock cycle for each level of sequential depth in the design, the entire sequential circuit can be put into a known, reproducible state. The controller then puts the circuit into the BIST mode where the initialized value in the circular BIST register is applied to the general sequential logic. The output value of each logic cone is then compacted and shifted one cycle for use as the next

input pattern to the circuit. As the BIST sequence continues, each bit propagates to the SAR whereby it undergoes normal data compaction associated with signature analysis. After a specified number of clock cycles (determined by the controller), the SAR is disabled from any further data compaction until the signature is read by ATE or by the maintenance processor used in system diagnostics.

Seven Circular BIST implementation examples are described below [Reference #3]. Each description contains information on chip area overhead, number of system flip-flops, number of system flip-flops which have been replaced by the BIST flip-flop, percentage of the device tested by the circular BIST technique, and fault coverage obtained on the portion of the device which is tested by circular BIST.

CASE HISTORY OF IMPLEMENTATION (#1):

Device #1 is a production device which has been fielded for three years and contains 1,820 system gates with 8K bits of on-board RAM.

1. Chip area overhead = 9.7%
2. Number of System Flip-flops = 152
3. Number of BIST Flip-flops = 92
4. Portion of chip tested = 100%
5. Effective chip fault coverage = 93.4%

CASE HISTORY OF IMPLEMENTATION (#2):

Production device in product manufacturing stage with 16,820 system gates.

1. Chip area overhead = 18.9%
2. Total Number of System Flip-flops = 920
3. Number of BIST Flip-flops = 739
4. Portion of chip tested = 97%
5. Effective chip fault coverage = 91.6%

CASE HISTORY OF IMPLEMENTATION (#3):

Production device in product manufacturing stage with 16,215 system gates.

1. Chip area overhead = 14.5%
2. Total Number of System Flip-flops = 983
3. Number of BIST Flip-flops = 576
4. Portion of chip tested = 93%
5. Effective chip fault coverage = 92.5%

CASE HISTORY OF IMPLEMENTATION (#4):

Production device in product manufacturing stage with 5,360 system gates.

1. Chip area overhead = 6.6%
2. Total Number of System Flip-flops = 248
3. Number of BIST Flip-flops = 72
4. Portion of chip tested = 39%
5. Effective chip fault coverage = 90.3%

CASE HISTORY OF IMPLEMENTATION (#5):

Production device in product manufacturing stage with 5,674 system gates and 2K bits of on-board RAM.

1. Chip area overhead = 13.6%
2. Total Number of System Flip-flops = 358
3. Number of BIST Flip-flops = 230
4. Portion of chip tested = 100%
5. Effective chip fault coverage = 91.0%

CASE HISTORY OF IMPLEMENTATION (#6):

Production device in product manufacturing stage with 13,937 system gates and 32K bits of on-board RAM.

1. Chip area overhead = 7.4%

2. Total Number of System Flip-flops = 784

3. Number of BIST Flip-flops = 575

4. Portion of chip tested = 99%

5. Effective chip fault coverage unknown at publication time

CASE HISTORY OF IMPLEMENTATION (#7):

Production device in product manufacturing stage with 15,394 system gates and 8K bits of on-board RAM.

1. Chip area overhead = 13.1%

2. Total Number of System Flip-flops = 790

3. Number of BIST Flip-flops = 656

4. Portion of chip tested = 99%

5. Effective chip fault coverage unknown at publication time

SIMILAR TECHNIQUE (#1): Circular Self-Test Path (CSTP)

SIMILAR TECHNIQUE (#2): Highly Integrated Logic Device Observer (HILDO) (See reference #6)

COMPLEMENTARY TECHNIQUE (#1):

A number of BIST techniques have been proposed for highly regular structures such as RAMs and PLAs which can be integrated with circular BIST. One such technique is explained in detail in Reference #5.

COMPLEMENTARY TECHNIQUE (#2):

Scan methods can be considered as a complementary technique since a partial scan path is implemented in the BIST register via the shift mode. By adding a serial input pin and an external mode pin, a complete scan capability can be easily accommodated with this technique.

GENERIC DESIGN PROCEDURES:

1. The first step necessary to implement the Circular BIST method in a design is to selectively replace the system flip-flops with the special BIST flip-flop of Figure 1.2 below. Further work on developing a near-optimal selection of flip-flops to be replaced is needed. However, the current approach is to replace those flip-flops whose combinational logic input cone exceeds a user-defined number of inputs such as three. The reasoning behind this approach is that flip-flops with small combinational logic input cones are assumed easily testable and can be readily tested by the outputs of the circular BIST chain.

2. Any BIST flip-flop elements which reside in the critical timing paths of the circuit can then be removed in order to reduce the impact of the BIST circuitry on the device performance.

3. Any BIST flip-flops which result in a register adjacency condition should be reconfigured or removed to reduce the chance of error masking.

4. Input multiplexers are added to isolate the system data from the circular BIST data in order to have reproducible results. Otherwise, the logic value sequences at the circuit primary inputs will affect the value of the signature in the SAR at the conclusion of the self-test sequence.

5. The BIST controller and SAR must be added to the circuit to control the self-test sequence and to store the compressed signature until accessed by system diagnostics or external ATE equipment.

6. Additional I/O pins are added for the BIST controller and the serial interface to the SAR.

LOGIC STRUCTURES USED OR REQUIRED:

System registers in the design are selectively replaced by the BIST Flip-Flop shown in Figure 1.2 whose operational modes are listed in Table 1.1.

These BIST flip-flops are connected in a circular path by chaining the output of one to the serial data input of the adjacent one. The last BIST flip output is connected to the serial data input of the first BIST flip-flop as shown earlier in Figure 1.1.

In the Circular BIST method, a Signature Analysis Register (SAR) is also included within the BIST chain and is strategically located to facilitate easy access during system diagnostics. The SAR stores the signature resulting from the self-test sequence until it can be read by the tester or the system test controller. In addition to the SAR, 2:1 multiplexers are required at the circuit inputs to effectively isolate the input system data from the BIST circuitry in order to obtain reproducible results. All of

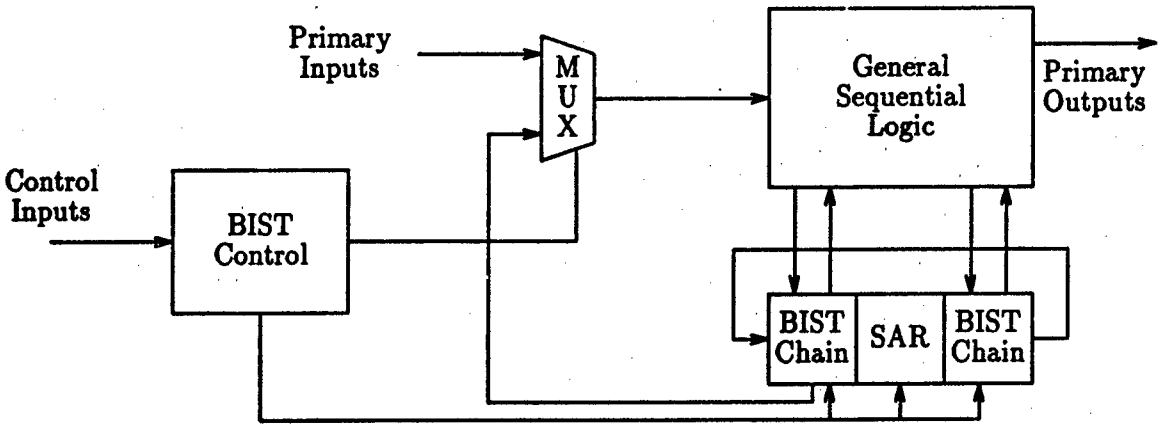

Figure 1.1. Architecture of the Circular BIST Technique

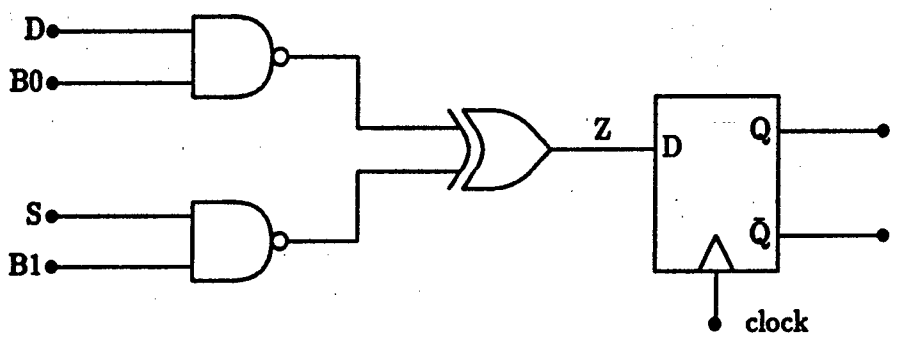

Figure 1.2. BIST Flip-Flop

B0	B1	Value at Z	Mode
0	0	0	Reset
0	1	S	Shift
1	0	D	System
1	1	D $\oplus$ S	BIST

Table 1.1. BIST Flip-Flop Operational Modes

this circuitry is controlled by a BIST control circuit which must be included on-board the device.

AVAILABLE TOOL (#1):

CONES - Behavioral Model Synthesis System developed at AT&T Bell Labs

AVAILABLE TOOL (#2):

CKT (Circuit Know Thyself) - CAD Tool developed at AT&T Engineering Research Center

EXTERNAL INTERFACE REQUIREMENTS:

Through I/O pins which control the operational mode sequence and provide access to the SAR.

INTERFACE TO ATE OR MAINTENANCE PROCESSORS:

The signature resulting from the self-test sequence needs to be provided to either the ATE or Maintenance Processor for comparison to the known good signature. Access to the signature can be provided through a serial interface.

APPLICABLE TO ON-LINE CONCURRENT TEST? No

APPLICABLE TO OFF-LINE CHIP TEST? Yes

APPLICABLE TO ENGINEERING TEST? Yes

APPLICABLE TO PRODUCTION TEST? Yes

APPLICABLE TO COMBINATIONAL LOGIC? No

APPLICABLE TO SEQUENTIAL LOGIC? Yes

APPLICABLE TO MEMORY (RAM/ROM)? No

SPECIFIC CIRCUIT TYPES TO WHICH IT IS APPLICABLE:

Particularly attractive for general sequential logic in that the Circular BIST technique can be readily automated.

IMPACT ON TEST PATTERN SET SIZE:

Compared to the test sets typically generated by the D-Algorithm (or other deterministic ATPG), the number of test patterns for the circular BIST technique is larger.

IMPACT ON TEST APPLICATION TIME:

In general, BIST techniques exhibit shorter test application times due to the at-speed generation and application of test patterns without the need for long shift sequences to scan in test vectors and scan out test responses as required for scan methods. The circular BIST technique seems especially attractive in that the test patterns are applied at-speed to the CUT. The required test time to generate the same number of different test patterns has been shown to be around 5% more than that of an ideal generator. One of the more attractive features of this method compared to other self-test methods is that the whole chip is tested in one session , whereas an exhaustively self-testing chip may require multiple test sessions in order to test each combinational block.

FAULT DETECTION COVERAGE:

Most of the reported results in using circular BIST indicate fault coverage levels in excess of 90% could readily be obtained.

DESIGN PENALTY INFORMATION:

1. Logic gate overhead in the circular BIST technique is roughly equivalent to that associated with scan methods. Reported results indicate an area overhead of 6-19% and a logic overhead of 6-28%. Compared to other self-test techniques such as BILBO which use multifunctional registers, The amount of overhead for circular BIST is smaller.

2. Some performance degradation is expected due to the extra logic required in the BIST registers unless measures are taken to remove BIST flip-flops from the critical paths in the design.

3. At least three additional I/O pins are required to configure the DUT into the proper operational mode (normal or test mode), and to access the resulting

signature. More additional I/O pins may be depending on the complexity of the self-test controller.

EFFECTIVENESS INFORMATION:

As with other data compression techniques such as signature analysis, some errors which may occur during the self-test operation may result in a signature identical to the good-circuit value. The classical model for signature analysis has an error escape probability of 2^{-k}, where k is the length of the SAR.

A number of simulation studies by Pilarski [Reference #2] and Stroud [Reference #1] have shown that after an initial setup time, the test patterns generated by the circular BIST chain exhibit good pseudorandom properties. However, the problem of register adjacency has been shown to cause fault masking in certain cases resulting in lower effectiveness of the technique.

Another problem which can limit the effectiveness of this technique is limit cycling. Limit cycling occurs when only a small subset of the pseudorandom patterns is generated before the circular path enters a repeating loop. In such cases, the effectiveness of the circular BIST technique is severely hampered by the low fault coverage which is obtained. It has been noted by Pradhan et. al. [Reference #4] that the probability of entering a loop during the test running time can be decreased by increasing the length n of the circular path. In circuits with only a few system flip-flops, the probability of limit cycling can be reduced by inserting extra BIST registers at the expense of increased overhead.

LITERATURE REFERENCES:

1. C.E. Stroud, "An Automated BIST Approach for General Sequential Logic Synthesis" Proc. 25th Design Automation Conference, 1988, pp.3-8.

2. A. Krasniewski and S. Pilarski, "Circular Self-Test Path: A Low Cost BIST Technique" Proc. 24th Design Automation Conference, 1987, pp.407-415.

3. C.E. Stroud, "Automated BIST for Sequential Logic Synthesis" IEEE Design and Test of Computers, December, 1988, pp.22-32.

4. M. Pradhan, E.J. O'Brien, S.L. Lam, and J. Beausang, "Circular BIST with Partial Scan" Proc. Int. Test Conf., 1988, pp.719-729.

5. S.K. Jain and C.E. Stroud, "Built-In Self-Testing of Embedded Memories" , IEEE Design and Test of Computers, Vol.3, No.5, October 1986, pp.27-37.

6. F.P. Beucler and M.J. Manner, "HILDO: The Highly Integrated Logic Device Observer" VLSI Design , CMP Publications, June 1984, pp.88-96.

2. STRUCTURED BIT TECHNIQUES

FULL NAME OF TECHNIQUE

Technique #2 - LSSD On-Chip Self-Test (LOCST)

DEFINITION OF TECHNIQUE

LOCST is an off-line BIT technique which provides both pseudorandom test generation and test response compaction at the chip-level and is based on the feedback shift register approach. This technique assumes that the device will be designed according to the Level-Sensitive Scan Design (LSSD) methodology. The input register is configured into a pseudorandom pattern generator (PRPG) which applies patterns to both the application logic and also provides patterns serially to the LSSD Shift Register Latches (SRL) configured in a scan chain. Similarly, the output registers are configured into a signature analyzer (SA) which compresses the output data stream into a unique signature.

DESCRIPTION OF TECHNIQUE

The LOCST technique complements the normal LSSD scan path testing methodology by implementing a pseudorandom-based self-test capability with only a nominal increase in overhead circuitry. The ideal implementation involves adding boundary scan registers to all primary inputs and outputs. These boundary SRLs are transparent during normal system operation but can act as shift registers, PRPGs, or SAs during test mode. If the overhead associated with boundary scan is too high, functional SRLs can be modified to support these self-test features. The LOCST architecture is shown below in Figure 2.1.

During self-test mode, the input SRL register is configured as a maximal-length pseudorandom pattern generator. An initial non-zero "seed" is placed into the PRPG and the internal SRLs and the output SA are initialized (often to the all zero state). Pseudorandom vectors are then serially shifted into the internal scan SRLs through the operation of the shift clocks. When the internal SRL register is filled, the functional clocks are activated for one cycle. In this manner, pseudorandom data is provided to the chip from the internal SRL register and from the PRPG which also feeds part of the application logic. When the internal SRL register is filled, the functional clocks are activated for one cycle. This captures the application logic responses into the internal register as well as the output register. However, the output register has been configured as a signature analyzer and thus compresses the parallel output response. Next, the shift clocks are cycled to completely shift out the internal SRL

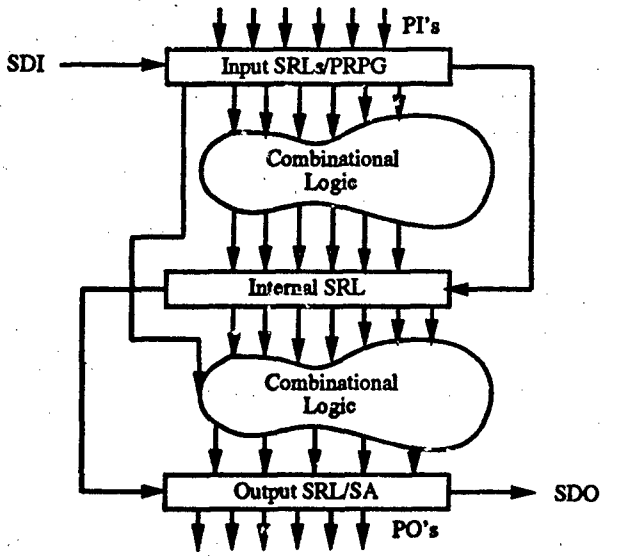

Figure 2.1. LOCST Architecture for an LSSD-based design

register into the signature analyzer. In this manner, the internal response of the application logic is serially compressed and mixed with the previously captured parallel response in the output SA. While this shifting is taking place, the next pseudorandom pattern is being serially shifted into the internal SRL from the PRPG. When the final response has been shifted into the SA, the new pseudorandom pattern has been shifted into the internal SRL register and the entire test cycle repeats. This cycle is repeated until the fault coverage reaches an acceptable level or until the test pattern cycle begins to repeat.

Three devices that incorporate the LOCST technique are described immediately below. Each of these devices performs a digital signal processing function. The overhead rate reported (<2%) does not account for the LSSD overhead or the overhead associated with the On-Chip Monitor (OCM), part of which controls the self-test logic. In addition, the interior logic fault coverage is the statistical fault coverage for the portion of the device tested by LOCST method with 95% confidence.

CASE HISTORY OF IMPLEMENTATION (#1)

1. Chip area overhead < 2.0%
2. Number of System SRLs = 213
3. Number of Random Patterns = 2000

4. Percentage of interior logic = 87.5%
5. Interior logic fault coverage = 97.3%
6. Test time (1MHz Scan clock rate) = 0.43 sec

CASE HISTORY OF IMPLEMENTATION (#2)

1. Chip area overhead < 2.0%
2. Total Number of System SRLs = 230
3. Number of Random Patterns = 500
4. Percentage of interior logic = 79.0%
5. Interior logic fault coverage = 97.5%
6. Test time (1MHz Scan clock rate) = 0.12 sec

CASE HISTORY OF IMPLEMENTATION (#3)

1. Chip area overhead < 2.0%
2. Total Number of System Flip-flops = 223
3. Number of Random Patterns = 3000
4. Percentage of interior logic = 85.0%
5. Interior logic fault coverage = 97.7%
6. Test time (1MHz Scan clock rate) = 0.67 sec

SIMILAR TECHNIQUE (#1)

Mixed-Mode Self Test (MMST) [Reference #5]

SIMILAR TECHNIQUE (#2)

Random Test Socket [Reference #4]

SIMILAR TECHNIQUE (#3)

Centralized Verification Testing (CVT) [Reference #6]

COMPLEMENTARY TECHNIQUE (#1)

LSSD Scan Test can be considered as a complementary technique to LOCST since a scan path is implemented in the DUT. Deterministic testing can still be performed according to the normal LSSD methodology of scanning in internal states, applying parallel test vectors, and scanning out the test responses.

GENERIC DESIGN PROCEDURES

1. The first step necessary to implement the LOCST method in a design is to add transparent boundary scan SRLs to the primary inputs and outputs. If a boundary scan capability is not to be incorporated into the design, then additional transparent SRLs must be added for those primary inputs which feed combinational logic blocks directly. These transparent input latches operate only in the test mode and are disabled during normal system operation. In similar fashion, transparent latches are added to primary outputs which are not directly fed by SRLs.

2. The input latches are then modified to provide a capability for generating pseudorandom test patterns. Multiplexers are added to isolate the internal latches from the input and output SRLs. The output SRLs are modified to support a serial and parallel signature analysis capability.

3. The OCM module is modified to control the self-test logic in addition to the LSSD scan test methodology that it already supports.

4. Additional I/O pins are added for the OCM inputs and outputs.

LOGIC STRUCTURES USED OR REQUIRED

Ideally, transparent boundary scan SRLs are added to each of the primary inputs and outputs for use in external testing of board interconnect and isolation of chip logic from other devices. These SRLs are also used to form the PRPG and SA during self-test mode. In addition to the basic SRL of the LSSD design methodology, a few additional multiplexers, feedback gates, and control logic are required to implement

the LOCST technique. The feedback gates are used in the input and output LFSRs for pseudorandom pattern generation and signature analysis. Some control logic may be needed to disable the data port system clocks in the input LFSR to isolate the system data from the self-test sequence in order to obtain a reproducible test. The multiplexers are needed to enable the feedback circuitry and isolate the internal scan path from the serial data input pin as shown below in Figure 2.2.

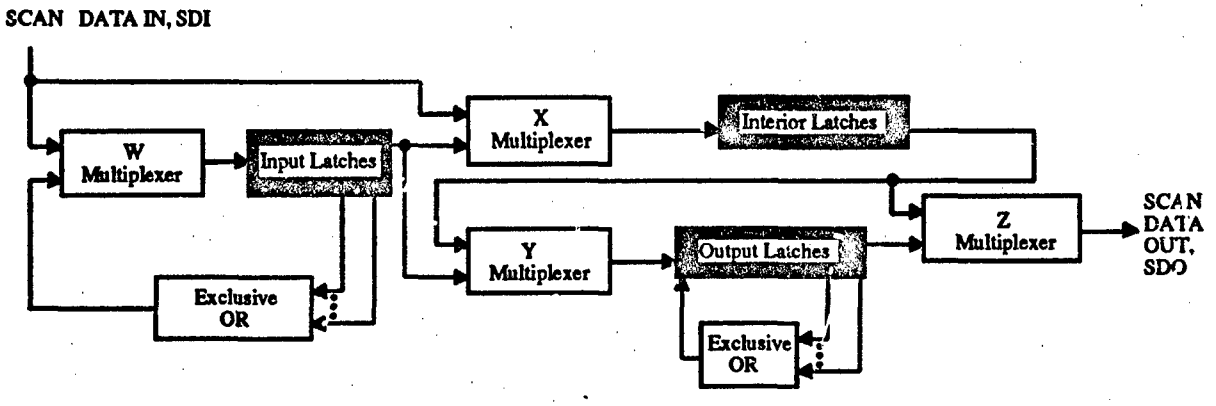

Figure 2.2. Arrangement of SRLs for LSSD Self-Test

AVAILABLE TOOL (#1):

No tools have been reported. However, the implementation of this technique for an LSSD-based design is a very simple overlay of the functional design. Tools do exist to convert a design into one that adheres to an LSSD design methodology.

EXTERNAL INTERFACE REQUIREMENTS

Through I/O pins which control the operational mode sequence and provide access to the shift register latches. In the LOCST implementation, an On-Chip Monitor (OCM) is included to control the test procedure (including the self-test circuitry). Eight additional I/O pins are required for testing purposes. According to reference #1, six of these are required for the self-test logic, most of these being SRL clock phases.

INTERFACE TO ATE OR MAINTENANCE PROCESSORS

The signature resulting from the self-test sequence needs to be provided to either the ATE or Maintenance Processor for comparison to the known good signature. Access to the signature is provided through a serial interface. The initial seed vector

for the PRPG is usually provided through an external serial interface although it is possible to store it on board the device. Careful control of the external clock sequence is required to perform this self-test technique.

APPLICABLE TO ON-LINE CONCURRENT TEST? No

APPLICABLE TO OFF-LINE CHIP TEST? Yes

APPLICABLE TO ENGINEERING TEST? Probably not, due to the compression of response data by the output signature analyzer. However, some research has been done whereby diagnosis of failures can be done using random pattern testing through the collection of additional response data [Reference #7].

APPLICABLE TO PRODUCTION TEST?

Probably not, due to the slow test application speed caused by the serial data transfer. In addition, the I/O logic is not tested by LOCST and requires some additional deterministic testing (easily done if boundary scan is included) for this type of logic. Since a much greater number of random patterns is needed to achieve the same fault coverage obtained in deterministic testing, this technique will require much longer test application times than normal deterministic LSSD testing. This technique is best suited for board and module level testing.

APPLICABLE TO COMBINATIONAL LOGIC? No

APPLICABLE TO SEQUENTIAL LOGIC? Yes

APPLICABLE TO MEMORY (RAM/ROM)? No

SPECIFIC CIRCUIT TYPES TO WHICH IT IS APPLICABLE

Particularly attractive for general sequential logic in that the LOCST Self-Test technique can be readily automated. For circuits with embedded RAM, additional techniques must be applied to the RAM circuitry since LOCST will not completely test these.

IMPACT ON TEST PATTERN SET SIZE

Since pseudorandom patterns are being generated while the serial shifting through

the SRL chain is taking place, large numbers of vectors are "wasted". A lower bound on the number of unique vectors is $(2^n - 1/m)$; where n is the length of the pseudorandom test pattern generator and m is the number of SRLs in the internal scan path. This effect where only a subset of all possible patterns is applied before the cycle begins to repeat is known as limit-cycling. The number of unique patterns which are applied to the DUT is

$$p = \text{Number of patterns} = LCM((2^n - 1), m)/m$$

where LCM represents the least common multiple of the two arguments. Thus, if the least common multiple of the two arguments is their product, then and only then can the full set of $2^n - 1$ patterns be applied. If there exits a least common multiple q, such that

$$q \neq m * 2^n - 1$$

then limit-cycling will occur after the p^{th} "good" pattern since the patterns will repeat. Limit cycling can be avoided by adding extra non-functional (dummy) SRLs to the internal scan chain such that the least common multiple of m and $2^n - 1$ is their product. These extra dummy SRLs could then be used as a transparent latch to sample key nodes that are difficult to observe during the exercising of the functional clocks in the self-test sequence.

IMPACT ON TEST APPLICATION TIME

Since this technique requires large amounts of serial shifting of pseudorandom patterns throughout the SRL chain, test times tend to be slow compared to other self-test methods. This method can support some at-speed testing of the application logic if the functional clock pulse occurs very soon after the test pattern has been shifted into place. Nevertheless, complete coverage of all delay faults is generally not possible with LOCST. The self-test time can be reduced if multiple scan paths are used which are loaded in parallel with data from the PRPG. Pseudorandom data can be fed to the multiple paths simultaneously from the PRPG or from different taps. The test responses can then be compacted using a parallel signature analyzer. Since multiple scan paths are supported, fewer shifts will be required to scan in each pseudorandom vector to the internal SRLs, thereby reducing the self-test time.

FAULT DETECTION COVERAGE

The reported results in using circular BIST indicate fault coverage levels in excess of 80% could readily be obtained with test pattern sets under 3000 vectors. The

relatively low total coverages result from the inability of the LOCST technique to test the I/O latch circuitry. It should be noted that coverages in excess of 95% were obtained using less than 3000 vectors on the portion of the devices which were being directly tested by the self-test circuitry. The overall coverage figures are moderated by the portions of the device not testable by the LOCST technique. However, if one includes boundary scan, very good coverages can be obtained by augmenting the self-test with a few functional patterns to test the chip boundary.

DESIGN PENALTY INFORMATION

1. Logic gate overhead in the LOCST technique is 2% above the overhead associated with the LSSD scan path. Total overhead including the LSSD and OCM control circuitry is estimated around 15% of the total logic. The boundary scan logic can also impact the overhead, although this logic could be incorporated in the I/O pads themselves, thus minimizing the core logic overhead.
2. At least six additional I/O pins are required to control the self-test procedure. Most of these are required for additional test clock phases.

EFFECTIVENESS INFORMATION

As with other data compression techniques such as signature analysis, some errors which may occur during the self-test operation may result in a signature identical to the good-circuit value. The classical model for signature analysis has an error escape probability of 2^{-k}, where k is the length of the SAR.

As mentioned earlier, most of the chip boundary logic as well as part of the OCM module are not tested by the LOCST method. By including some additional functional tests for the boundary scan logic, a high level of coverage seems plausible. The problems associated with the LOCST technique are the long self-test time required and the difficulty of obtaining good delay fault coverage. Also, the overhead can be fairly significant if one accounts for the LSSD overhead in addition to the self-test circuitry.

In terms of automation, the LOCST technique can be easily incorporated into a LSSD-based design. Since the VENUS system already supports the LSSD design methodology, this technique should be fairly easy to incorporate into the VENUS system. This technique seems fairly attractive in that it has low overhead (when designing with the LSSD methodology) and provided good overall coverage of a device when supplemented by boundary scan tests.

LITERATURE REFERENCES

1. D. Komonytsky, "LSI Self-Test Using Level Sensitive Scan Design and Signature Analysis", Proc. 1982 International Test Conference, pp.414-424.

2. J.J. LeBlanc, "LOCST: A Built-In Self-Test Technique", IEEE Design and Test of Computers, November, 1984, pp.45-52.

3. D. Komonytsky, "Synthesis of techniques creates complete system self-test", Electronics, March 10, 1983, pp.110-115.

4. P. Bardell, W.H. McAnney, J. Savir, *Built-In Test for VLSI: Pseudorandom Techniques*, John Wiley and Sons, Inc., 1987.

5. H.H. Butt and Y.M. El-Ziq, "Impact of Mixed-Mode Self-Test on Life-Cycle Cost of VLSI Based Designs", Proc. 1984 International Test Conference, pp.338-347.

6. D.L. Liu and E.J. McCluskey, "High Fault Coverage Self-Test Structures for CMOS ICs", Proc. 1987 Custom Integrated Circuits Conference, pp.68-71.

7. F. Motika et al., "An LSSD Pseudo Random Pattern Test System" , Proc. 1983 International Test Conference, pp.283-288.

3. STRUCTURED BIT TECHNIQUES

FULL NAME OF TECHNIQUE:

Technique #3 - CrossCheck

DEFINITION OF TECHNIQUE:

CrossCheck is currently being applied to enhance the testability, automatic test pattern generation, and fault coverage of a gate array design. The base array is predesigned and manufactured by the chip manufacturer to include a grid of probe lines and sense lines and the associated test controller. The gate or cell is also predesigned to incorporate a small sampling transistor. When the logic design is completed, the user's design is superimposed on the base array. Figure 3.1 shows the user's design superimposed on the base array. In this arrangement the gate of the sampling transistor is connected to the probe line, the drain to the sense line, and the source to the output of the gate or cell. Figure 3.2 shows a more detail schematics of the gates (or cells), probe lines, sense lines, and their connections. In normal operation, the sampling transistor is switched off and the gate output is disconnected from the sense line. In testing mode, the sampling transistor is switched on by the active probe line and connects the gate output to the sense line. Thus, the crosspoint switches at the intersections of probe lines and sense lines provide a large number of test points (or check points) that can be accessed or observed externally. This nearly 100 per cent observability is the major contribution to the enhanced testability in the CrossCheck technique. The software, part of the CrossCheck methodology, offers the ASIC designer controllability by taking advantage of the high observability to reduce the search space and to speed up (reportedly 40 times faster) the test generation and fault simulation processes. The speed up is due to the fact that faults are observed at the point of their occurrence and do not need to be propagated to the outputs.

Currently, CrossCheck does not implement Built-In-Self-Test (BIST) technique. But CrossCheck is compatible with the proposed JTAG test bus standard and therefore can incorporate BIST and other scan testing techniques.

DESCRIPTION OF TECHNIQUE:

The key structure of the CrossCheck test is a grid of test points designed and built into an ASIC's base array by the chip's manufacturer. The test points are formed and addressed through probe lines and sense lines. For a base array with 100 probe lines and 100 sense lines, there are 100x100=10,000 test points that are externally accessible. The access to the test points is controlled by the test controller cell through the probe line driver and the sense line receiver. The linear feedback

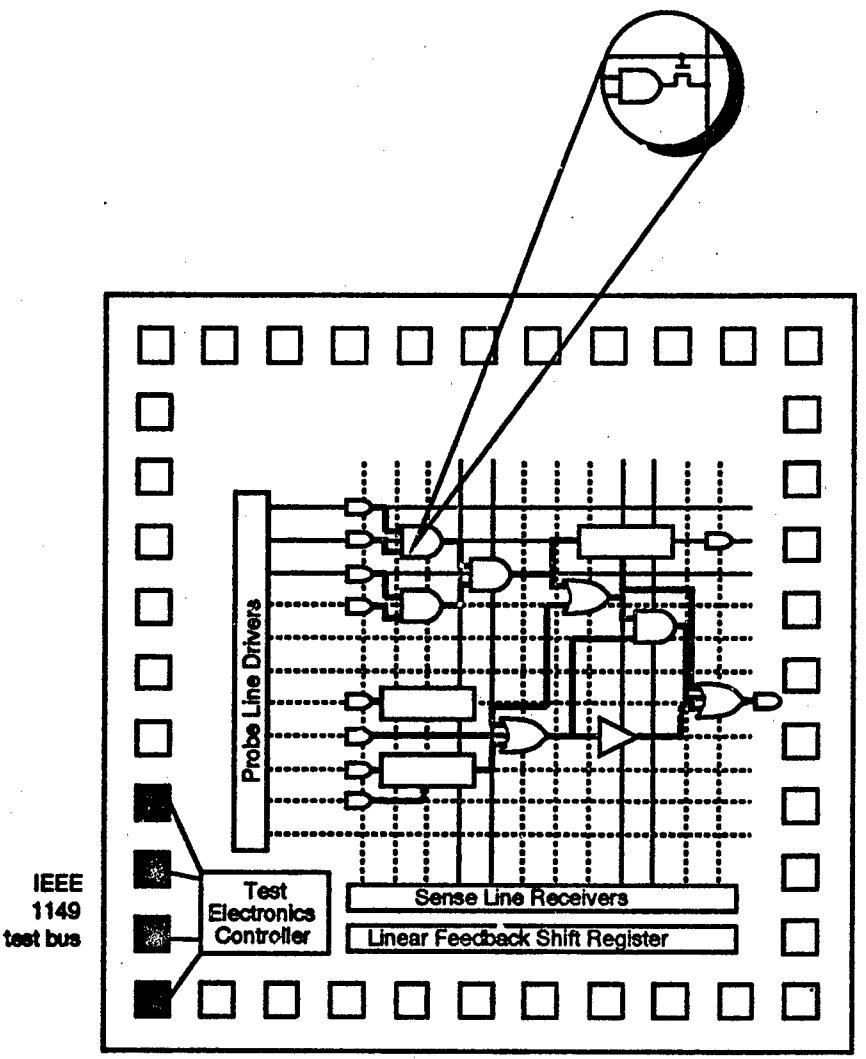

Figure 3.1. Logic Design Superimposed on the Base Array

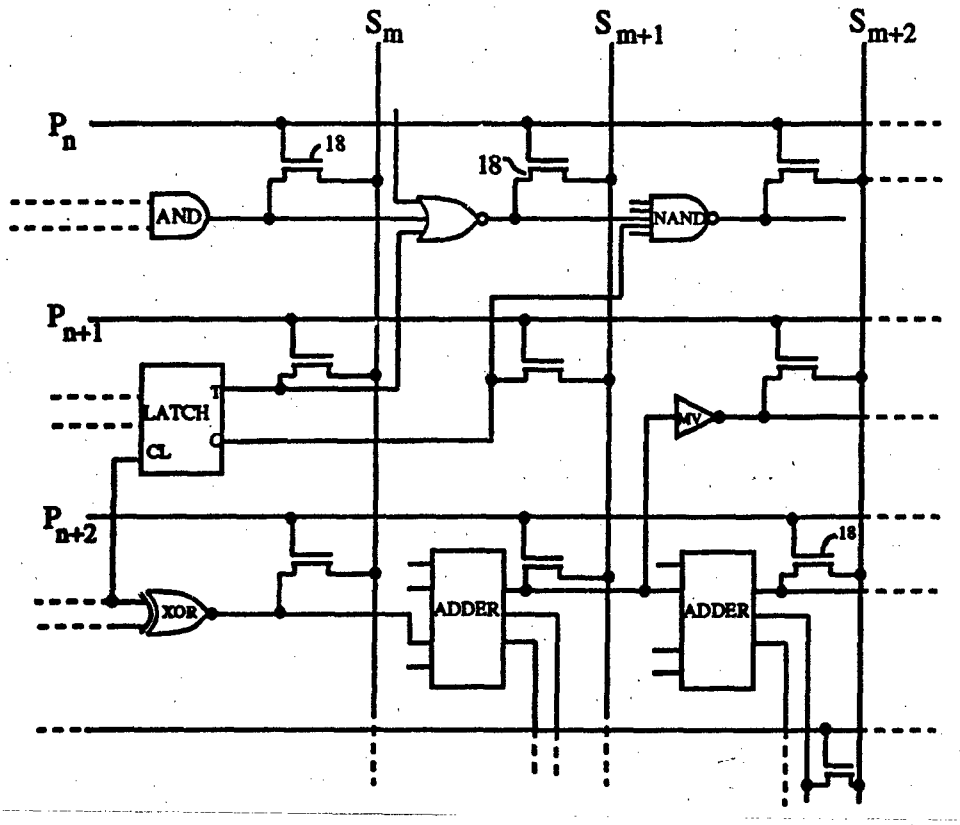

Figure 3.2. Gates and Cells with Switching Transistors and Connections

Pn, Pn+1, Pn+2: probe lines
Sm, Sm+1, Sm+2: sense lines

shift register (LFSR) may also be used to compress the test outputs received from the sense line receiver. The four (or optionally five) I/O pins on the chip provide access to the test controller which is designed to be compatible with the proposed IEEE 1149.1 (JTAG) testbus standards.

The user develops his logic design as usual. The design is then placed and routed in such a way that when the ASIC is fabricated, the design is automatically superimposed onto the base array which embeds the probe structure as described above. The gate or cell in the CrossCheck technology is predesigned to include a very small transistor at the cell's output end to provide connections between the cell and the probe line and the sense line.

The source of the transistor is connected to the output of the cell, the drain to the sense line, and the gate to the probe line. The output of the cell is transferred, through the transistor, to the sense line when the transistor is on which is triggered by the active probe line. Thus the small transistor serves as a switch that connects or disconnects a cell's output to the sense lines depending on whether the probe line is active or inactive. In normal operation, the switch transistor is switched off to disconnect the cell from the associated sense line. In test mode the transistor is switched on to allow the value of the associated node to be read onto the sense line. The GO/NO-GO testing can be performed by capturing the response of the circuit in the form of a signature and then reading out the signature through the TDO pin. Signature generation involves selecting the probe line in the gate array one at a time, loading the sense line values into the signature register, and then clocking the signature register.

Similarly, the value of any test point can be read out through the TDO pin when performing diagnostic access. CrossCheck is compatible with the proposed JTAG test bus standard and can accommodate BIST or other scan techniques. Figure 3.3 shows the CrossCheck test architecture which is similar to the JTAG test architecture. The important difference is the provision of test point array, probe line driver, and the sense line receivers in the CrossCheck.

Since the crosspoint switching transistors are integrated with the logic cells of the ASIC, they do not cause additional routing capacitance to the design. They do add a small overlap capacitance which is about 1 per cent for a typical fan-in load at the output of the logic gate. The increase in overall capacitance, including wiring capacitance, is 0.2 per cent for a typical design with a fan-out of two. Thus, the small additional capacitance has a negligible effect on propagation delay and the system performance. The area overhead due to test points and the associated test control logic is about 10 per cent on a 10,000-gate gate array ASIC. For a larger gate array, the proportion of the area overhead will be smaller.

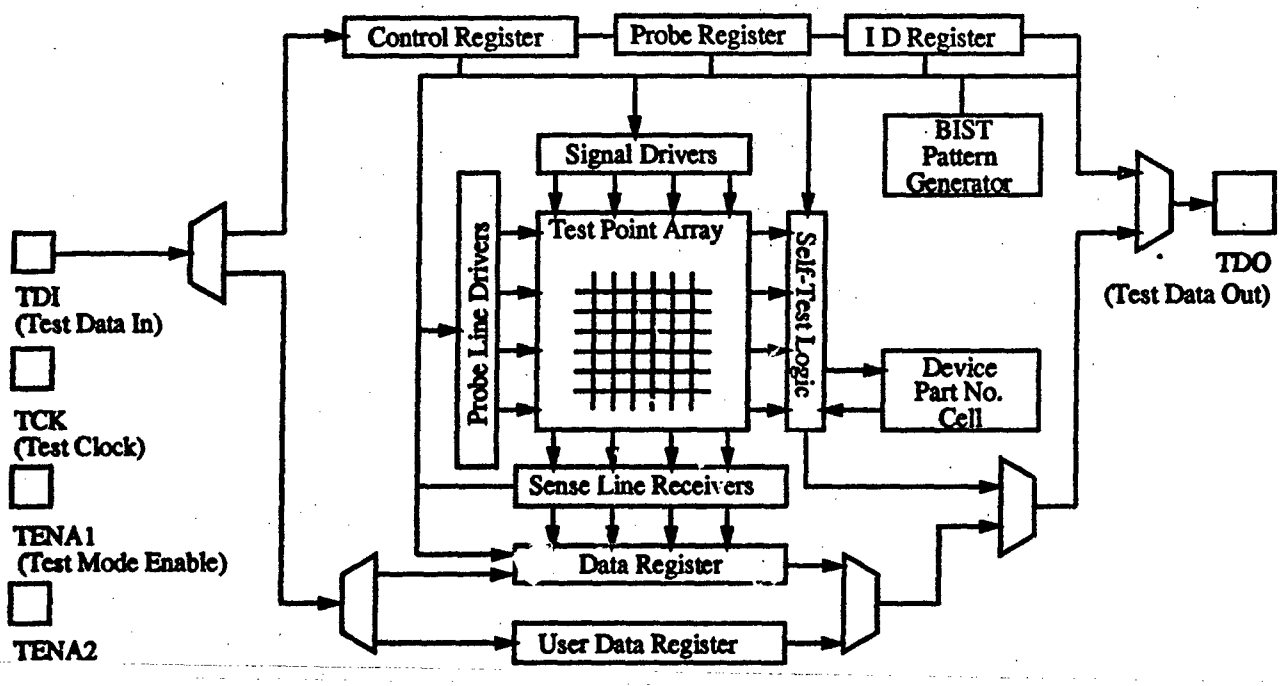

Figure 3.3. CrossCheck Test Structure

The average fault coverage for ten ISCAS combinational circuits ranging from six gates to 3568 gates was about 99.6 percent with 1000 pseudo-random test patterns. For benchmarks with two 10,000-gate CMOS gate array, Circuit1 and Circuit2, the CrossCheck technique showed significantly higher fault coverage than the non-CrossCheck methods. These designs contained deeply buried sequential logic, complex decoders driven from buried instruction registers, 16 and 24 bit counters, and control state machine. For Circuit1, the CrossCheck achieved 99.6 percent fault coverage while the conventional technique only reached 72.8 percent fault coverage. The total test generation and fault simulation time on a SUN3/160 was about 4 hours for CrossCheck and about 146 hours for conventional technique. For Circuit2, the CrossCheck had 99.8 percent fault coverage and the conventional technique had 92.5 percent fault coverage.

TESTING METHODOLOGY

The large number of built-in test points will result in nearly 100 per cent observability. The software is then applied to improve the controllability. The test pattern automatically generated by the software can be used to supplement the original functional patterns created by the designer. The first step of the testing is to initialize the circuit to a stable and known state. The fault detection is then proceeded in the following steps:

1. Run fault simulation with the user-developed functional test patterns. If higher fault coverage is required, then supplement the functional vectors with the vectors automatically generated by the CrossCheck. Alternatively, the user can choose to generate all the test vectors automatically using CrossCheck.

2. CrossCheck can detect stuck-at fault and other faults. To detect faults that may not be detected by the stuck-at fault model, apply a complementary signal on the test point via sense line and make an analog measurement of the signal amplitude on the test point. The non-stuck-at faults such as bridging faults, internal shorts and opens usually cause the device to operate out of its noise margin.

3. To prepare for ATE testing, expand the test pattern set into low level test bus patterns. The signature values missing from the expansion can be read out from the TDO pin. This involves selecting the probe lines one at a time, loading the sense line values into the signature register (MISR), and then clocking the signature register.

4. Add the missing TDO values to the test patterns before ATE translation can begin.

5. Perform the GO/NO-GO testing to screen out bad chips. The user may also run diagnostic testing to determine process-related problems.

If the scan or BIT structure is incorporated in the ASIC, then the procedures for running the scan or BIT can be applied.

SIMILAR TECHNIQUE (#1):

Test Point Placement to simplify Fault Detection [Reference #5]

GENERIC DESIGN PROCEDURES:

1. In CrossCheck, cells (or gates) are modified to include a small sampling transistor. Also needed are a modified base array that consists of probe lines, sense lines, probe line driver, sense line driver, and the test controller along with extra test pins TMS, TCK, TDI, and TDO. These are all predesigned and manufactured by the chip manufacturer. Therefore the designer proceeds the logic design as usual. The designer is not required to specifically implement certain design-for-testability structures.
2. If the CrossCheck is to incorporate boundary or BIT technique, then the design procedures for these techniques should be applied.

LOGIC STRUCTURES USED OR REQUIRED:

The gate or cell needs to be modified to incorporate a small sampling transistor as shown in Figure 3.1. A base array that provides a grid of probe lines and sense lines, a probe line driver, a sense line receiver, a test controller with four test pins are also required in the CrossCheck technique.

AVAILABLE TOOL:

The tools for implementing the CrossCheck methodology is available from CrossCheck Technology, Inc. and possibly from LSI Logic Corporation.

EXTERNAL INTERFACE REQUIREMENTS:

Four extra pins are needed for the CrossCheck technique: TMS (Test Mode), TCK (Test Clock), TDI (Test Data In), and TDO (Test Data Out).

INTERFACE TO ATE OR MAINTENANCE PROCESSORS:

The test pattern set, which consists of test patterns and high level test instructions, must be expanded into low level test bus patterns. The signature values missing from the expansion can be read out from the TDO pin and added to the test patterns before ATE translation can begin.

APPLICABLE TO ON-LINE CONCURRENT TEST?

NO.

APPLICABLE TO OFF-LINE CHIP TEST?

Yes

APPLICABLE TO ENGINEERING TEST?

Yes

APPLICABLE TO PRODUCTION TEST?

Yes

APPLICABLE TO COMBINATIONAL LOGIC?

Yes

APPLICABLE TO SEQUENTIAL LOGIC?

Yes

APPLICABLE TO MEMORY (RAM/ROM)?

No

SPECIFIC CIRCUIT TYPES TO WHICH IT IS APPLICABLE:

Synchronous digital circuits, especially gate array and standard cell designs.

IMPACT ON TEST PATTERN SET SIZE:

The test pattern generation process proceeds with pseudo-random patterns, followed by weighted pseudo-random patterns, and then finally deterministic patterns. The weighted pseudo-random pattern is based on the back-tracking of undetected faults. The size of the test pattern set generated by the CrossCheck is usually smaller

than other pseudo-random techniques because each test pattern can detect many faults due to the high observability, smaller search space, and the shorter path for propagating fault effects. However, due to its pseudo-random nature, the test pattern size in CrossCheck is larger than that of a user-developed functional test patterns. But it takes much longer time to develop a high fault coverage test patterns manually.

IMPACT ON TEST APPLICATION TIME:

The fault simulation is very fast since the fault structures do not have to be propagated through complex blocks of logic to an observable point.

FAULT DETECTION COVERAGE:

The fault coverage of 98 to 99 percent can be easily achieved. This is because the CrossCheck technique uses not only the stuck-at fault model but also other fault models such as transistor opens and shorts, net opens, and net bridging to uncover those faults that are not detected by the stuck-at approach.

DESIGN PENALTY INFORMATION:

Since the crosspoint switching transistors are integrated with the logic cells of the ASIC, they do not cause additional routing capacitance to the design. They do add a small overlap capacitance which is about 1 per cent for a typical fan-in load at the output of the logic gate. The increase in overall capacitance, including wiring capacitance, is 0.2 per cent for a typical design with a fan-out of two. Thus, the small additional capacitance has a negligible effect on propagation delay and the system performance. The area overhead due to the grid of test points and the associated test control logic is about 10 per cent on a 10,000-gate gate array ASIC. For a larger gate array, the proportion of the area overhead will be smaller.

Four extra pins per chip are needed for test mode control, test clock, test data input, and test data output.

EFFECTIVENESS INFORMATION:

CrossCheck provides an attractive and efficient approach to the problems of ASIC testability and automatic test pattern generation. The CrossCheck technology is currently being applied by LSI Logic to CMOS channelless gate-array design. According to the literature, CrossCheck can also be applied to channeled gate-array, standard cell, or even custom design styles, though possibly with more area penalty. The concept is also applicable to other process types including BiCMOS, and Bipolar. Due mainly to the nearly 100 per cent observability, the technology can also greatly

facilitate the testing of boards and systems. References indicated that CrossCheck, with negligible speed penality and about 10 per cent area overhead, is less painful to implement than scan or BIST techniques. It is worth noting that CrossCheck is compatible with other design-for-testability techniques such as scan, BIST, and the proposed JTAG standard and uses the same four test pins, namely TMS, TCK, TDI, and TDO. One of the literature references indicated that CMOS dynamic logic can not be tested with the CrossCheck technique. If this is the case, alternative designs may be worked out to incorporate the CrossCheck method.

LITERATURE REFERENCES:

1. T. Gheewala, "CrossCheck: A Cell Based VLSI Testability Solution", 26th ACM/IEEE Design Automation Conference, 1989, pp.706-709.

2. G. Swan, Y. Trivedi, D. Wharton, "CrossCheck- A Practical Solution for ASIC Testability", The Proceedings of the International Test Conference, 1989, pp.903-908.

3. M. Carroll, "Built-In Array Payoff - Better Fault Detection", High Performance Systems, August, 1989, pp.32-44.

4. T. Gheewala, Grid-based, "CrossCheck" Test Structure For Testing Integrated Circuits, U.S. Patent 4,749,947, June 7, 1988.

5. J. P. Hayes et al, "Test Point Placement to Simplify Fault Detection", IEEE Symposium on Fault Tolerant Computing, 1973.

*U.S. GOVERNMENT PRINTING OFFICE: 1993-710-093-60162

MISSION

OF

ROME LABORATORY

Rome Laboratory plans and executes an interdisciplinary program in research, development, test, and technology transition in support of Air Force Command, Control, Communications and Intelligence (C^3I) activities for all Air Force platforms. It also executes selected acquisition programs in several areas of expertise. Technical and engineering support within areas of competence is provided to ESD Program Offices (POs) and other ESD elements to perform effective acquisition of C^3I systems. In addition, Rome Laboratory's technology supports other AFSC Product Divisions, the Air Force user community, and other DOD and non-DOD agencies. Rome Laboratory maintains technical competence and research programs in areas including, but not limited to, communications, command and control, battle management, intelligence information processing, computational sciences and software producibility, wide area surveillance/sensors, signal processing, solid state sciences, photonics, electromagnetic technology, superconductivity, and electronic reliability/maintainability and testability.